Escuela
Politécnica
Superior

A Framework to Generate and Label Synthetic/Real Video Data to Feed Temporal Segment Networks

Bachelor´s Degree in Computer Engineering

Bachelor´s Thesis

Author:
Daniel Finn Allhoff
Supervisors:
José García Rodríguez
John Alejadro Castro Vargas

May, 2020

Universitat d'Alacant
Universidad de Alicante

A Framework to Generate and Label Synthetic/Real Video Data to Feed Temporal Segment Networks

Author
Daniel Finn Allhoff

Supervisors
José García Rodríguez
Departamento de Tecnología Informática y Computación
John Alejadro Castro Vargas
Departamento de Tecnología Informática y Computación

Bachelor´s Degree in Computer Engineering

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, May 2020

Abstract

In this project, we propose an action prediction and a data generation pipeline. While, the former makes use of Deep Learning, the latter results in a pipeline that makes possible the generation of real and synthetic data. Moreover, to feed the deep learning method a large amount of annotated data is needed. For this purpose an action tagging tool is also featured. Furthermore, in order to supply the lack of data, we have also proposed a video data augmentation pipeline for action recognition purposes. While the 3DPLab team developed a photorealistic synthetic data generator called UnrealRox, we will use this system working with some sequences recorded with a mocap to generate the necessary synthetic data. We have generated a total of 5 different useful sequences with a complex setup of 3 kinects and a capture motion suit. Finally, we have deployed and tested the novel Temporal Segment Network with the state of the art Action Recognition dataset UCF-101.

Resumen

En este proyecto, proponemos el desarrollo de un trabajo de investigación en los campos de la predicción de acciones y la generación de datos. Mientras que la propuesta para resolver el primer problema hace uso del Aprendizaje Profundo, el segundo desarrollo permite la generación tanto de datos sintéticos como datos reales. Además, dado que los algoritmos de Aprendizaje Profundo requiren de una gran cantidad de datos, hemos desarrollado un aumentado de datos dirigido a problemas de reconocimiento de acciones. El equipo 3DPLab ha desarrollado un generador de datos sínteticos fotorealistas denominado UnrealRox. Hemos usado este sistema con secuencias generadas por un sistema de captura del movimiento e para poder generar suficientes datos sintéticos. Específicamente, se han generado un total de 5 secuencias distintas mediante un complejo despliegue de 3 kinects y un traje de captura de movimiento. Finalmente, se ha desplegado y testeado la red neuronal Temporal Segment Network con una base de datos del estado de arte del reconocimiento de acciones llamada UCF-101.

Acknowledgements

This work would have not been possible without the help of the 3DPerceptionLab Team. In the beginning, when I have chosen this topic as Bachelor´s Thesis I did not know If I would afford it. However, It has been possible and, now I know that I would repeat the same choice over and over.

Firstly, I would like to thank José García and John Castro, my supervisors. Jose García has been kindly guiding me and supporting me in each decision. John Castro has patiently helped me in each problem in the development of this Thesis.

Moreover, I want to thank Pablo Martinez-Gonzalez, Sergiu Oprea and Albert Garcia who also helped helped a lot in Thesis.

Last but not least, I would like to thank my family. I want to thank them because they have funded me the university even when they did not were bounded to. Moreover, they have been there every time supporting me. I am proud of them, they have showed me to not surrender, to have courage and do not step back.

Persistence is very important.
You should not give up unless you are forced to give up

Elon Musk.

Contents

List of Figures

List of Acronyms

2D	Two-Dimensional.
2D CNN	2D Convolutional Neural Network.
3D	Three-Dimensional.
3D CNN	3D Convolutional Neural Network.
BOVW	Bag of Visual Words.
BVH	BioVision Hierarchy.
CNN	Convolutional Neural Network.
CUDA	Compute Unified Device Architecture.
DL	Deep Learning.
DTW	Dynamic Time Warping.
GWR	Growing When Required Network.
HMDB-51	The Human Motion Database is a dataset with 51 classes.
HOF	Histogram of Optical Flow.
HOG	Histogram of Oriented Gradients.
JSON	JavaScript Object Notation.
LSTM	Long Short-Term Memory Neural Network.
MBH	Motion Boundary Histogram.
OS	Operative System.
RANSAC	Random sample consensus.
ROS	Robot Operating System.
STIPS	Space-Time Interest Points.
SURF	Speeded-Up Robust Features.
TSN	Temporal Segment Network.
UE	Unreal Engine.
UFC-101	The Human Actions Database is a dataset with 101 classes.
UML	Unified Modeling Language.

1. Introduction

This first chapter introduces the main topic of this work and it is organized as follows. Firstly, Section 1.1 introduces the framework developed in this thesis. Next, Section 1.2 presents the motivation of this work. Section 1.3 defines the outcome of this work. Section 1.4 mentions the final goal of the project. Finally, Section 1.5 shows the planning we followed.

1.1. Overview

Today's society faces a wide variety of social challenges. The care of the dependent population, a phenomenon that affects all ages, is one of the most relevant. A clear example is the care and rehabilitation of people with congenital or acquired disabilities caused by traffic, work or domestic accidents, which represent a significant number of dependent persons. The deterioration of physical and cognitive abilities limits the autonomous life of people. This is an issue in which developed countries have put major interest. In the particular case of Spanish society, demographic forecasts indicate that, in 2020, there will be around 1.8 million dependent persons in Spain, an amount that will be increased due to the population aging phenomenon. In this sense, there is a growing interest in the analysis and study of how new technologies can help improving the living conditions of dependent persons. The conclusion is that it is necessary to provide personalized assistance to dependent persons. This work is part of project under development in the 3DPLab at the Institute or Computing Research (IUII) where it is proposed to build a system able to identify changes in the normal behavior of persons that may be useful to detect anomalous situations (tremors, disorientation, memory loss, etc.) or to help early disease detection in apparently healthy people.

In this context, the result of this project will be applied to estimate the patient's intention when carrying out an action to help him finishing it. We propose the use of the information obtained through a monitoring system, augmented by VR/AR techniques, and processed with deep learning methods to identify the different people, objects, and interaction between them, which over time define a set of behaviors. We will segment each one of these behaviors in a number of primitive micro-actions. Defining the cycle of primitive actions and identifying which behavior the person is developing or intends to develop (using the context of the environment and previous actions) will enable us to estimate the action the person intends to perform and help her carry it out. Some help to finish the action or giving extra information about the person (for example name and relation with the patient) or object to interact with (for example if we are grabbing an empty cup in the kitchen and we are close to the coffee machine) will be provided to the patient in the event of detecting any indication of memory failure or any other type of inability to carry it out.

In this setting, we have particularly researched the task of action prediction with a deep learning-oriented approach. The main goal is to obtain an efficient enough pipeline that given video input images predicts the next action accurately. For this purpose, we have made use

of the Temporal Segment NetworkTSN [24]. The best parameter combination is searched for the best results. Moreover, the majority of the work has been related to the training data generation. We created tools to tag videos in a hierarchy of atomic actions, basic tasks, and behaviours. Several videos of daily actions were recorded using a motion capture system to integrate them in a virtual reality system generating a large amount of annotated data.

1.2. Motivation

This document is the result of the work done at the University of Alicante between the years 2019 and 2020, and represents the Bachelor´s Thesis emerged from the Computing Engineering Degree.

One motivation of this project is the collaboration with the *3D Perception Lab (3DPL)*, mainly focused on the Deep Learning (DL) field. On one hand, the team focuses on DL-based computer vision applied to robotics. On the other hand, due to the high performance needed for obtaining good computer vision results, they also push the limits in GPU computing.

This work is in line with the Spanish national project entitled *Monitoring and Detection of human behaviours for personalized Assistance and early disease detection (MoDeAsS)*, funded by *Ministerio de Ciencia, Innovación y Universidades* with Jose Garcia-Rodriguez and Miguel Angel Cazorla Quevedo as main researchers.

Finally, the project also comes with a strong fascination of what is possible to accomplish with the use of deep learning. The impact that DL can produce and already is making arouses great interest. Although the mathematics behind seems to be simple, the result in some tasks is at human level or even better.

1.3. Proposal

In this project, we propose an action prediction and a data generation pipeline. While the former makes use of Deep Learning, the latter results in a pipeline that makes possible the generation of real and synthetic data. Moreover, to feed the deep learning method a large amount of annotated data is needed. For this purpose, an action tagging tool is also featured [1]. The 3DPLab team developed a photorealistic synthetic data generator called UnrealRox therefore we will use this system working with some sequences recorded with a motion capture setup to generate the necessary data.

1.4. Goals

The main goal is to develop a system capable of performing action recognition and, as result, predicting the next possible actions. For accomplishing this goal, a study of the current deep learning methods applied to action recognition, and their related datasets or data generation systems will be carried out.

[1] All the tools and frameworks developed in this Thesis are available in `https://drive.google.com/drive/folders/1NcHgnui8gX8gwmdL0o-82GqOFyf-BdqS?usp=sharing`

As a main application, the system can be used to assist people with cognitive impairment, such as Alzheimer's disease. With time the neuronal damage gets worse, resulting in an abnormal behaviour as dementia hindering daily life.

Moreover, another goal is the development of a framework to tag videos for action classification purposes. This is related to the aforementioned synthetic data generation pipeline. This last goal aims to create of a big amount of action tagged video data. This will require the recording of real data with cameras and motion capture data simultaneously, which will be processed and used as own data.

Finally, with all the data, the goal is to apply the self created data and real data on the Temporal Segment Network and check the performance.

1.5. Planning

The proposed goals are developed during a timespan of 9 months, starting day 10th of October and ending the 26th of May. Throughout the development of the project it is considered as basis a period of 3 hours work a day. In total, approximately 15 hours work per week.

The initial planning is distributed into a total of four main tasks. Each task is divided into specific ones. As the starting phase, **Action Manual Data Tagger**, it consists of developing a fully video data tagging tool. It has the next subtasks:

1. **Documentation**
2. **Create interface in QT**
3. **Add image tagging compatibility**
4. **Add video tagging compatibility**
5. **Prepare BVH tool** for adding BVH animation tagging compatibility
6. **Integration** with the main QT tool
7. **Debugging**

As the second phase, **Real Data Creation** (obtaining the real data). It has the next subtasks:

1. **Documentation**
2. **Preparing software** for the recording
3. **Setup** of the environment
4. **Testing**
5. **Recording**

As the third phase, **Synthetic Data Creation**, whose main goal is obtaining synthetic data. It has the next subtasks:

1. **Virtual Environment preparation**

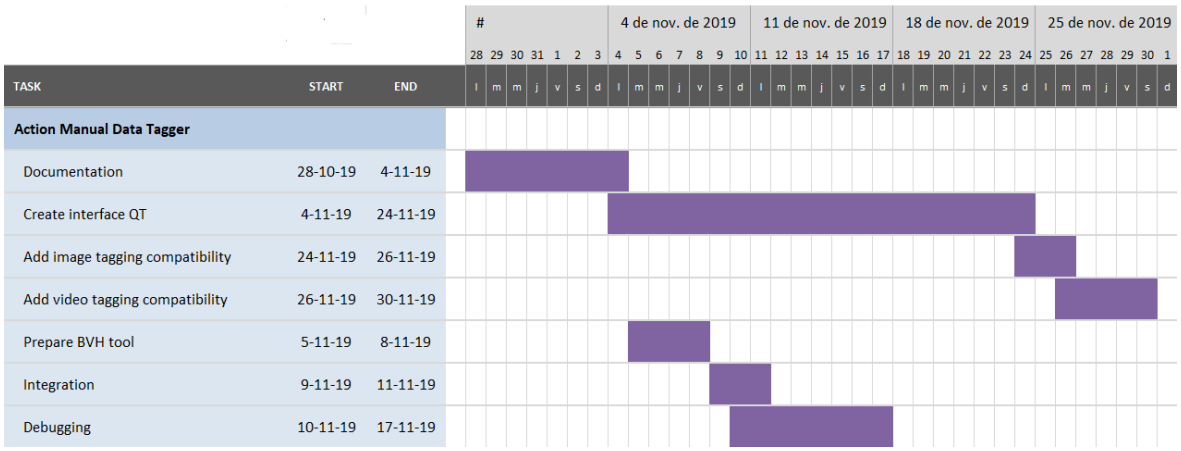

Figure 1.1: Task 1 Gantt diagram.

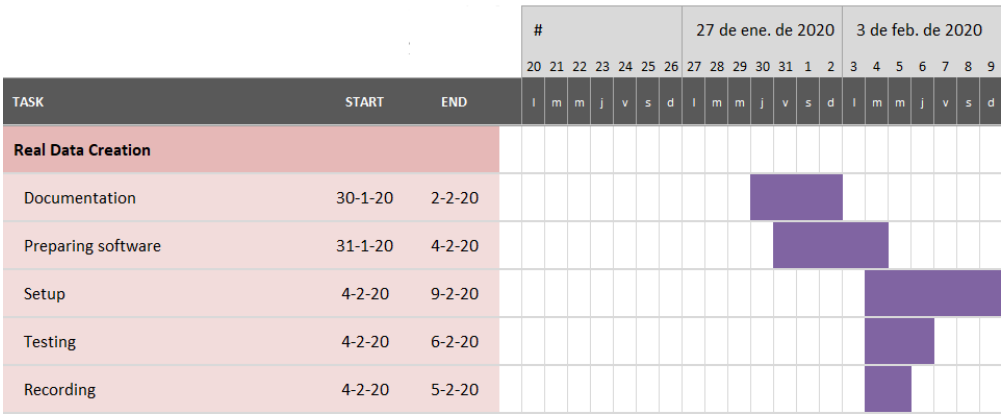

Figure 1.2: Task 2 Gantt diagram.

2. **Preparing data processing pipeline** like data augmentation
3. **Data generation**
4. **Data tagging** with the initial tool

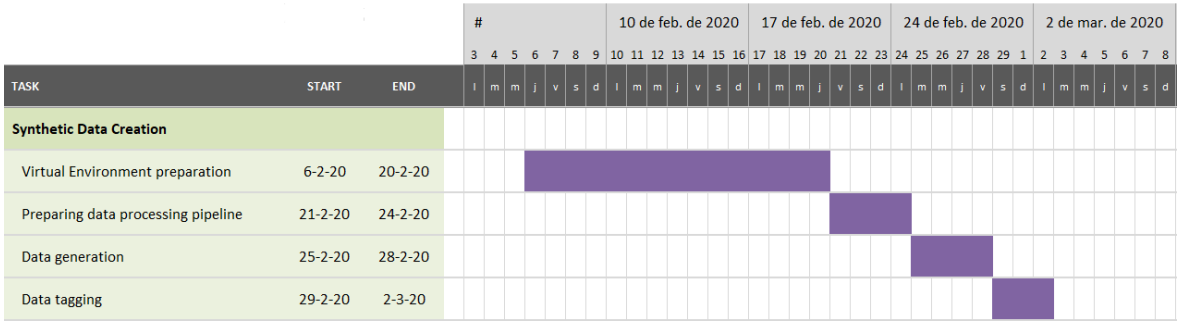

Figure 1.3: Task 3 Gantt diagram.

The last phase, **Experimentation with Deep Learning**, is the task of applying the generated data to the data-driven architecture. It has the next subtasks:

1. **Prepare environment**
2. **Optical flow extraction** of the data
3. **Temporal Segment Network Training**
4. **Experimentation**

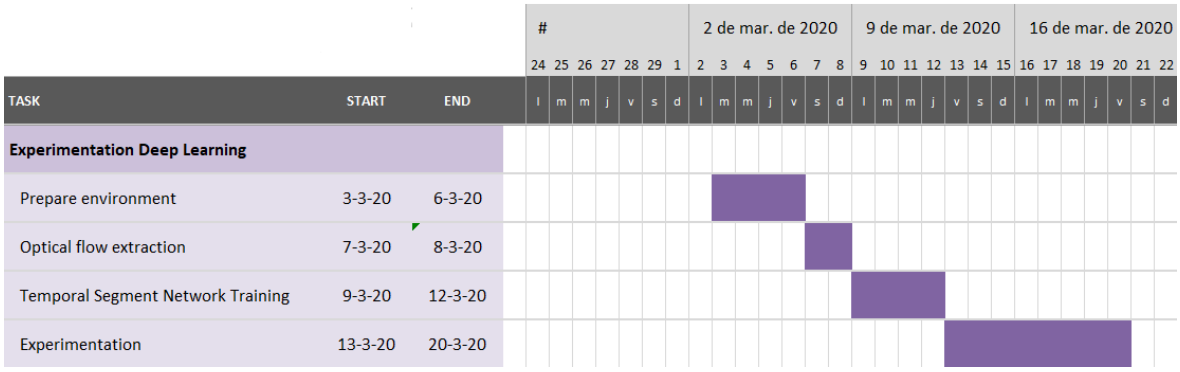

Figure 1.4: Task 4 Gantt diagram.

1.6. Outline

This document is structured as follows: Chapter 1 introduced the project, its motivations, and the main goals to accomplish as well as its distribution in the calendar. Then, in Chapter 2, we describe the state of the art in action recognition databases and generators; we review the best deep learning architectures as well. Chapter 3 describes the hardware and software tools that we will use. Following this, in Chapter 4, we detail all the development of a generation pipeline and a detailed description of the Temporal Segment Network. Finally,

in Chapter 5, the main conclusions of the Thesis are laid down as well as future lines of research.

2. State of the art

This chapter presents the state of the art of action recognition and prediction for human behaviour analysis. Moreover, datasets and data generation related systems are also reviewed. Section (2.1) describes the most important challenges in the action recognition and interpretation tasks. Section (2.2) presents some relevant datasets and data generation systems. Section (2.3) exhibits the main deep neuronal network architectures used.

2.1. Challenges

A lot of work has been done in the action recognition field. Remarkable problems have been tackled and accomplished to solve by using deep learning techniques. However, some up-to-date challenges degrades the performance of those algorithms.

As it is known, the more variation exists in the data, the harder is to learn from that data. This variation can be intra-class and inter-class. When the same action can be fulfilled with various motion styles, it is said that the data has an intra-class variation. We, as humans, are not perfect machines. We do the same action differently, and we cannot reproduce the same action perfectly. Moreover, the recording of the action inherently creates intra-class variation due to changes in the perspective and appearance. Furthermore, different actions can have a similar movement (inter-class), thus making the discrimination process a lot harder.

The action recognition or prediction is done over videos, made up by a lot of frames. Not all frames are needed due to it´s redundancy, little to none change occurs between frames making them almost identical. Moreover, some frames have small contribution to the prediction.

Deep learning architectures training stage need a lot of data to learn relevant data features. In supervised problems, as the action recognition/prediction problem, the data must be labeled. While a huge quantity of data is available from different sources, not enough of them are labeled. This implies the lack of datasets with multiple points of view and variations, which make them not suitable for a learning that generalizes well.

2.2. Data generation and Datasets

Neural network architectures and in general supervised learning methods need a lot of data to work with. This necessity has motivated that many research groups spent a lot of time to generate huge datasets to feed machine and deep learning algorithms in many different fields. In particular image an video datasets for different tasks like: objects segmentation, localization or detection, video classification and analysis, anomalous behaviour detection, action recognition and many others.

As creating this data manually is not feasible, synthetic data creation is a promising alternative. The advantages are: huge amount of data, reduced data generation time with reduced cost, ...etc. However, it has some disadvantages: the need of a realistic environment,

realistic behaviors, time of development in the environment, difficulty of applying it to the real world...etc.

Disregarding the disadvantages, the creation of synthetic data has many benefits. Moreover, there are many virtual environments and datasets that aim to make the generation of artificial data possible.

UnrealRox [11] is a virtual environment developed in Unreal Engine. This work benefits from the high photorealism that Unreal Engine achieves. The system provides offline recording in order to reproduce the movements from multiple points of view and to generate data in RGB, depth, segmentation and instance segmentation format. Moreover, it supports the use of Virtual Reality hardware where a novel realistic grasping is provided with the use of VR controllers. They also have released a dataset called Robotrix [6] developed using this virtual environment.

Gibson [26] is another virtual environment. First difference is that it does not create artificial environments, it virtualizes real spaces. Furthermore, the gap between generated and real world is decreased using a neuronal network that makes that conversion. Hence, this data can be used for training. This mechanism that makes this possible is called "Goggles". Moreover, the data that is already provided as database is huge. In conclusion, this virtual environment provides a great adaptation of the robots trained in artificial environments to the real world. It also has work on making navigation feasible, being a minor contribution.

Matterport3D [3] makes possible the perfect virtualization of a real environment (home environments) using a specific camera (Matterport Pro Camera). Thus, later, it is possible to simulate it giving a realistic result. As a supplementary material, it has been released a large-scale dataset of indoor scenes. The data contained is in RGB, with depth and semantic available information. As well, 360 and human-height with multiple points of view data is offered. This dataset has helped a lot in tasks of computer vision as being the most complete one.

Replica [20] is a dataset composed of photo-realistic reconstructions of house interiors in 3D. The highlight of this dataset is the high quality of the scenes it contains: high resolution of the models and textures, including glass and mirror information. In addition, semantic and instance segmentation is also provided.

Multimodal Indoor Simulator(MINOS) [15] is a simulator created with the finality of testing sensorimotor control models. MINOS is also used as a benchmark for indoor navigation algorithms. This environment provides support to two datasets: SUNCG (synthetic furnished houses) and Matterport3D (reconstructed real buildings). Diverse sensory inputs are supported too: vision, depth, surface normal per pixel, contact forces...etc. This helps get a closer sense of reality. This environment has made possible a lot of experiments: navigation algorithms, use of more sensory input than just visual and different complexities of the environment.

All the systems already proposed do not have an action simulation compatibility. At this point, the work focuses in behaviour analysis, mainly in action prediction. That´s why it is needed the most realistic simulation possible, where a subject is doing diverse actions in an environment. Then, using deep learning, behaviour analysis can be done automatically.

Cornell House Agent Learning EnvironmenT (CHALET) [27] is a 3D house simulation used for autonomous agents interaction. It contains default rooms and house configurations, but can be personalized.

The agents can be commanded and told to do certain actions using text, like moving objects and interacting with them.

One drawback is the realism, the scenes are really far from reality. Thus making it a bad learning environment for applying it directly on real world. Therefore, working on the 3D rendering quality is important. However, this environment provides an interesting functionality.

VirtualHome [14] is a virtual environment where household activities are simulated. It is programmed on Unity3D. The behaviors are specified with the use of programs. There is a list of actions and micro actions to perform by the agent. The project proposes the generation of programs from a list of natural language described tasks and the development of those tasks, in order to be recorded. This project is the most advanced in behaviour simulations.

The video obtained is in RGB-D format, with also the possibility of segmentation and pose segmentation. Also, a VirtualHome Activity Dataset was released.

The House of inteRactions (THOR) [8] main focus is the first person view from a human body. It has default configurations with objects inside that can be move in the scene. The environment makes feasible agent navigation, object interaction(moving and interaction) and actions list to do.

It´s photorealism is not remarkable. However, the easy framework handling of complex actions and realism object interaction with states and physics make this environment useful.

Habitat [16] is a highly photorealistic 3D simulator developed by Facebook. It provides the simulation of virtual robots making it the most realistic possible. In detail, the simulation uses AI allowing a realistic navigation in the environment, instruction following as "go somewhere or various places", question answering...etc.

It has a 3D dataset handling with built-in support for MatterPort3D, Gibson, Replica...etc

Habitat consists in: habitat-sim and habitat-api. The first one makes possible the 3D simulation and the second one is a modular library used for all the IA needed.

The functionality that makes this framework distinguished of the rest is the high standardization, generalization and realism accomplished. All this with maintaining high framerate.

They demonstrate that a navigation through learning gives better results than using Simultaneous Localization and Mapping (SLAM). Also, they have proven that only agents with depth sensors generalize well between other datasets.

The Human Motion Database (The Human Motion Database is a dataset with 51 classes (HMDB-51)) [9] is a database with 51 action categories. They are tagged with high-level annotations and actions. It has the problem that the data is described handmade. As result it is prone to error and, also, does not have pixel-level annotations. Moreover, the clips do not have a micro action level tagging which could be useful for more detailed predictions.

The Human Actions Database is a dataset with 101 classes (UFC-101) [18] is a 101 human actions classes dataset, being a extension of the UCF50. It is known as the largest dataset of human actions released. That´s why it´s the most challenging one, due to the high number of classes. However, it has the same drawback that the (HMDB-51) dataset.

Procedural Human Action Videos (PHAV) [19] is a procedural generative model for human actions data creation. The data obtained is diverse, being the most interesting the semantic segmentation of human bodies. They make use of motion capture sequences, programmed animations and, finally, both with ragdoll physics. This is a mixed approach: the movements are manually created, but then extended automatically with virtual simulations.

They propose the CoolTSN architecture, as well.

2.3. Architectures

When obtained the data, by using the correct architectures, it is possible to solve a lot of computer vision tasks. There are three main approaches: handcrafted, single stream networks and multiple stream networks.

2.3.1. Handcrafted features method with HOF, HOG and trajectories

These methods are based on handcrafting features and working on top with an algorithm that makes sense of the relevant information extracted. These features are called descriptors. The majority of handcrafted features methods in the action prediction field make use of optical flows or similar techniques that obtains the apparent motion information from videos.

Firstly, Histogram of Optical Flow (HOF) makes the distribution of all the orientations of the optical flow results. To compute the optical flow, a sequence of images is needed. It calculates the motion between frames. The result is the extraction of useful temporal information.

The most popular method to extract descriptors is the use of Histogram of Oriented Gradients (HOG). It works by counting the total gradients occurrences in portions of the image (Figure 2.1). Specifically, a distribution of the gradients directions is generated. Normally, it is rounded to 8 directions. Moreover, a normalization process is done for making it invariant to changes of illumination. The disadvantage is that it just obtains spatial information. However, it can work jointly with other methods for getting also temporal data. This seems to give beneficial results.

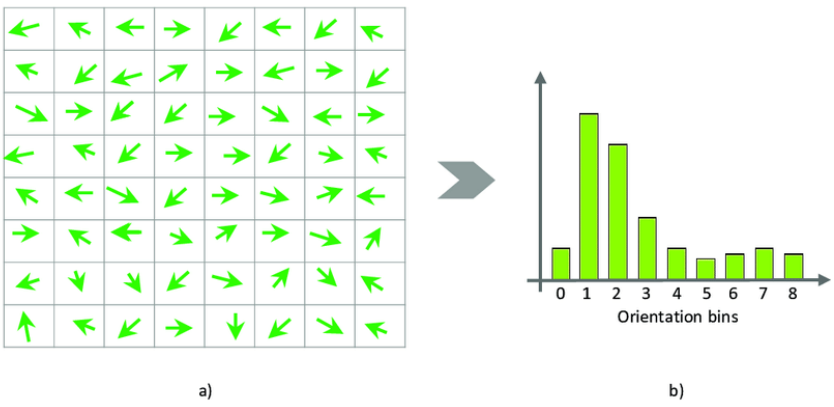

Figure 2.1: Conversion from gradients to histogram [1]

One more advanced method for feature extraction on single frames is the use of Speeded-Up Robust Features (SURF). Mainly, it uses the HOG. However, it's speed and efficiency relies on the use of integral images for summarizing the image. Moreover, the feature detection is made with box filters and size variation.

Up to this point, the feature extractors shown are used on single frames and not considering the temporal dimension of the videos. There exists feature extractors that take into account spatial and temporal information.

Space-Time Interest Points (STIPS) [10] is a feature extractor that can be used for representing video data. The interesting part is that does not use optical flow. They extract locations where images values have big spatial and temporal changes in both directions. The noise is taken account by comparing results with the neighborhood. The input data is Three-Dimensional (3D), being the third dimension the spatial domain.

The first one is the Motion Boundary Histogram (MBH) [4] which proposes a method that helps the camera motion removal. It consists of the calculation of the HOG depending on the two optical flow components: vertical and horizontal. This camera motion removal makes a robustness superiority over optical flows.

A generalization for videos of typical feature extractors are Dense Trajectories [22]. They calculate local motion in the videos. It works by densely sampling interest points and keeping track of the trajectory made by them, using optical flow fields (Figure 2.2). This sampling is done on diverse spatial scale and tracked on each of them separately. The result is a set of features that provide the direction of optical movement. A problem appears due to the use of optical flows when sudden actions occur, the tracking fails and new interest points need to be defined.

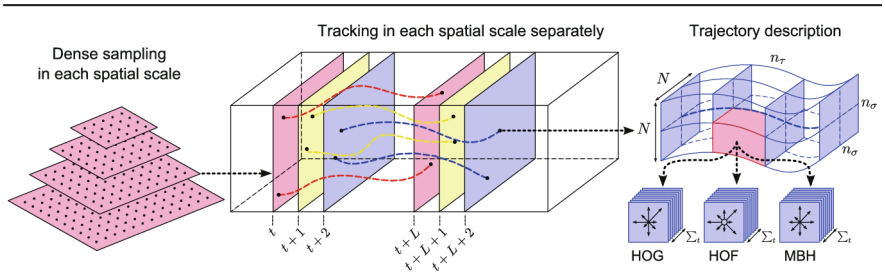

Figure 2.2: Dense trajectories pipeline extracted from [22]

A work branched from the Dense Trajectories resulted in the Improved Trajectories [23], obtaining more consistent trajectories. The upgrade version consider the camera motion. It works by estimating the motion of the camera and removing trajectories that follows this movement. Using SURF it matches feature descriptors between frames and then the homography is obtained with Random sample consensus (RANSAC). As a result, consistent dense trajectories are removed from this homography view, being these ones the camera motion.

The Bag of Visual Words (BOVW) [13] is a framework that makes possible the action classification/prediction with use of multiple descriptors that are fused (Figure 2.3). It uses feature extractors (dimensionality reduction and extraction of the relevant information) like STIPS and Dense Trajectories. Also, descriptors like HOG, HOF and MBH return points of interest that work on top the feature extractors.

This BOVW-based architecture use videos as frames that are not related in any order. This breaks the meaning that one image is before another, giving simplified information. As result, the processing of the information is faster and easier. However, due to its simplicity,

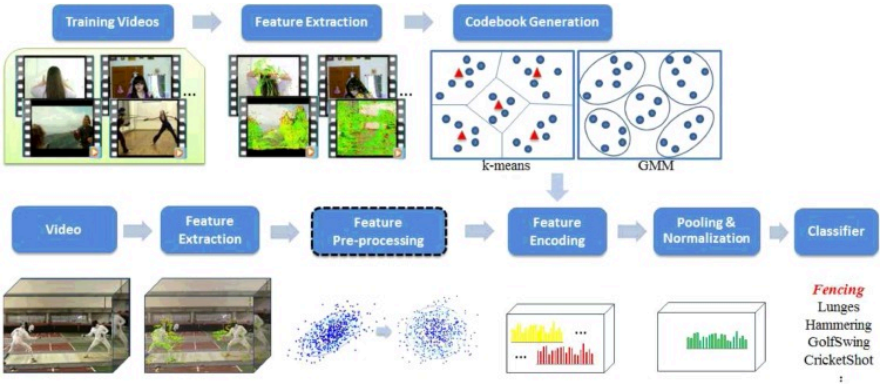

Figure 2.3: BOVW steps [13]

performance is degraded respect other methods.

Finally, all these feature extractors results in the use of machine learning techniques that predicts the class. Moreover, some features are useful for Deep Learning Architectures.

2.3.2. Single Stream

Single Stream methods works with deep neural networks that learns the relevant information to lookup for itself. The handcrafting of features it´s relevant due to it´s fast prediction. However, the natural complexity of the videos makes them unapproachable with high precision with the already mentioned methods.

At the beginning, 2D Convolutional Neural Network (2D CNN) karpathy2014 were an option for video classification. They are very good in spatial recognition. However, due to the impossibility to learn temporal changes it is needed to add more information that relates motion with time.

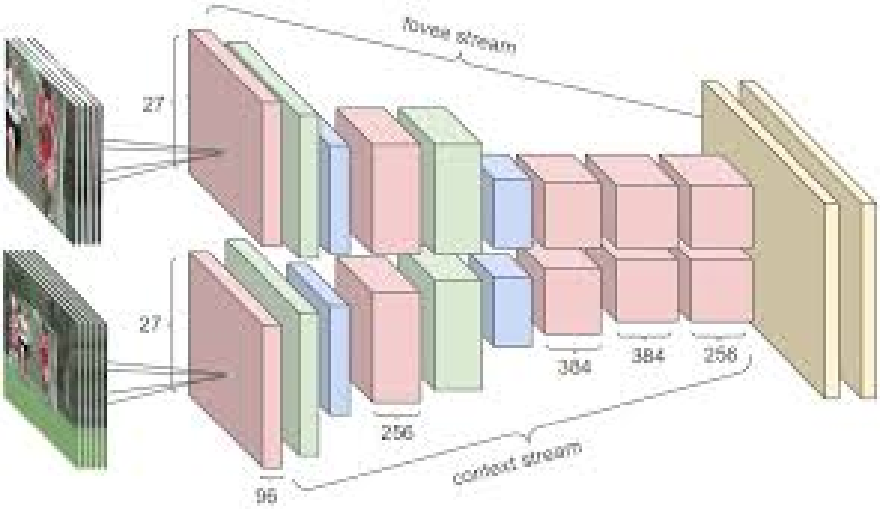

Figure 2.4: CNN action recognition oriented [7]

The 2D CNN with the Long Short-Term Memory Neural Network (LSTM) appeared due to the already high performance achieved by the 2D CNN on images. It makes use of the temporal pooling. The LSTM model is put on top of 2D CNNs, making it a sort of hybrid network. The resulting architecture, gets the advantage of both Convolutional Neural Network (CNN) and LSTMs, handling images and learning long-range information. However, this step has not the high-performance results needed, as other state-of-the-art networks has.

As alternative, when handling with consecutive frames, a simple 3D convolution is a viable option. It is also called C3D [21]. However, the 2D CNNs cannot be reused on 3D Convolutional Neural Network (3D CNN) and the network has a lot of parameters to learn (more than needed). Moreover, often it just learns with a image flow of 16 frames (due to learn restrictions), being not suitable for learning long-term information.

A totally different approach is using the Growing When Required Network (GWR) [12]. As in the paper states, is basing the behaviour analysis with a hierarchy of GWR networks, growing when the data cannot be represented properly by the existing nodes in the network. The actions and behavior are naturally displaced in a hierarchy: one behavior is made up from a combination of smaller actions. As result, this seems to give good results. It learns the information correctly and obtains good generalization in human-object interactions.

2.3.3. Multi Stream

Multi-Stream networks use multiple CNN to model space and motion information during time in videos. This has resulted in state-of-the-art architectures.

The first multi stream architecture that solves the spatio-temporal representation with the use of two 2D Convolutional Neuronal Networks [17] is the two stream 2D CNN. First One captures single frames representing the space and the second one CNN learns temporal changes between the frames. Each stream is classifying on it´s own, softmax normally. They are connected with the use of late fusion.

The temporal changes are successfully learned based on the use of a stack of optical flows. Optical flow is a technique that converts frames into a new frame that represents the change within those frames. This helps avoid the need of using a 3D CNN. Therefore, the complexity is lower, making faster learning times and better generalization. While learning, on each pass forward, two images(spatial frame and temporal frame) are passed and validated with which action is done at that moment or a prediction.

Two Stream Network fusion is almost identical to the Two Stream 2D CNN. However, as [5] proposed, the fusion mechanism that is done at the last layer (late fusion), is done before and late, inclusive (Figure 2.5). This helps to create a more robust pipeline with early information to both streams.

As the two stream architecture of 2D CNNs, the two stream 3D CNN [25] makes use of two neural networks: a 3D CNN for spatial information and second one 3D CNN for temporal information. In this case, it is a generalization of the 2D method for applying it to 3D. Late fusion is applied as well.

Nevertheless, due to the 3D CNN complexity, techniques must be used to avoid over-fitting. For example, dropout helps a lot in this case. Moreover, the two streams are pretrained separately.

Temporal Segment Network [24] appeared due to short context learning problems that are presented with the architectures already presented. Mainly due the use of optical flow and

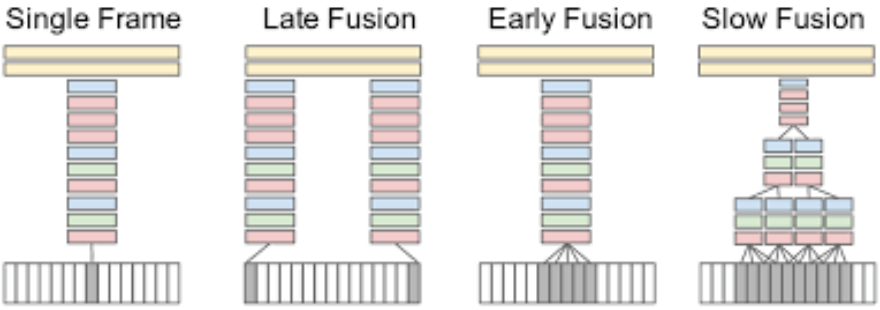

Figure 2.5: Fusion mechanisms extracted from [7]

small windows. The idea is to get a wide view of the video and remove frames that might be redundant. In videos with a high frame rate, it is not required to look up all the frames, because the majority of them has little to nothing change. The way it works is the next: the whole video is divided randomly in K segments with equal length. From each segment, a snippet is randomly extracted: spatial and temporal information. This is passed through it´s respectives 2D CNN and Temporal CNN). Finally, all scores obtained are late fused into one score of spatial information and one of temporal information (Figure 2.6).

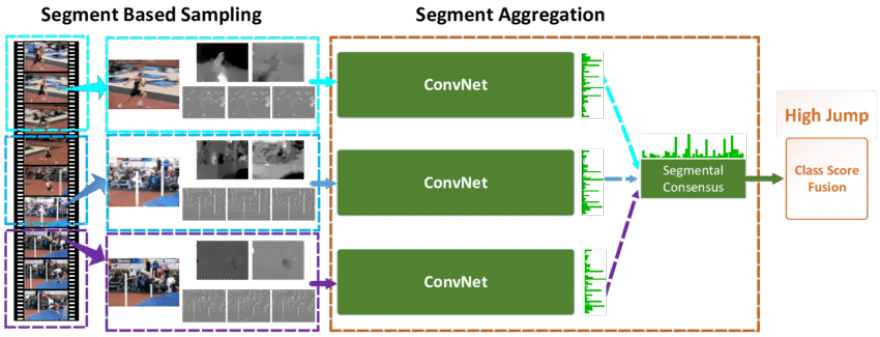

Figure 2.6: Temporal Segment Network [24]

The CoolTSN, proposed in the PHAV dataset [19], is a Temporal Segment Network trained on the combination of synthetic data and real data (Figure 2.7). On training, each one type of data has it´s own segmental consensus, having fully connected and softmax loss layer. The result is a multi-task loss of both losses.

The real data is obtained from the UCF-101 and HMDB-51 dataset, while the synthetic is created procedurally the most realistic possible by it´s PHAV environment. This is helpful on an better generalization and transfer to real world tasks.

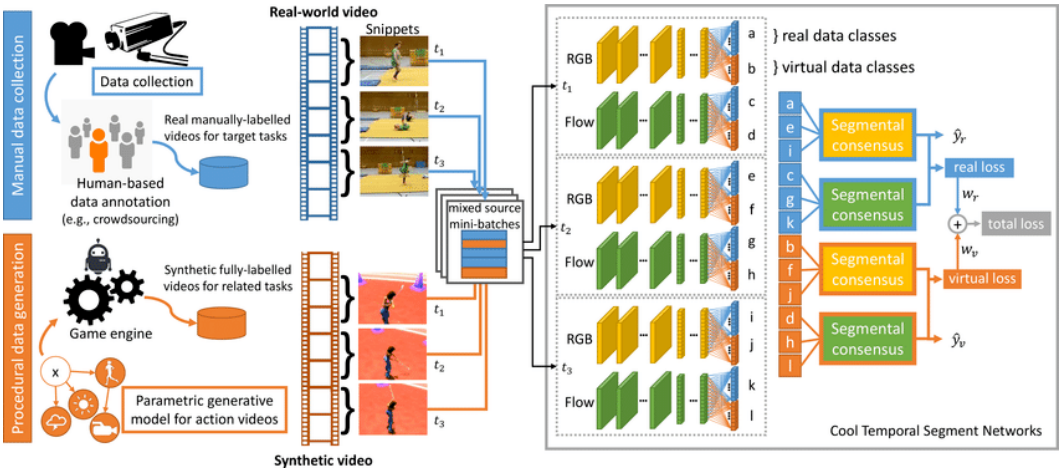

Figure 2.7: Cool Temporal Segment Network: Improved for synthetic data training [19]

Finally, Inflated 3D ConvNets (I3D) [2] is the state-of-the-art in action prediction and recognition. The concept of inflation refers to adding dimensions to the filters and pooling layers of a convolution neuronal network. Specifically, pre-trained 2D CNN are inflated for obtaining the resulting I3D.

3. Methodology

In this chapter the software and hardware used for accomplishing this Bachelor´s Thesis are presented. Section 3.1 describes Deep Learning frameworks used in this project. Section 3.2 presents software used for other kind of tasks throughout the development. Finally, Section 3.3 shows the hardware available in the 3DPLab and employed to train deep models.

3.1. Deep Learning Libraries and Frameworks

With the advent of deep learning, many software frameworks were developed and released to increase the accessibility and ease development of deep models. Each one of them (libraries and frameworks) offer different levels of abstraction, flexibility, and performance. A consequence of that is the fact that many backgrounds are covered: from theoretical experts in deep learning, cross-disciplinary researchers, newcomers, or engineers.

In this section, we will review three of the most common deep learning frameworks: TensorFlow, Keras, and PyTorch. Furthermore, we will also talk about a platform that powers them to achieve great efficiency: CUDA.

3.1.1. TensorFlow

One library that benefits on working on top of CUDA, is TensorFlow. It is a framework dedicated for Deep Learning tasks. It makes possible of a medium level of neural network creation and training. For this purpose, TensorFlow is used as standard. It can be used for creating new architectures. However, the flexibility is reduced mostly due the hard debugging when the architecture fails. The main processing data are tensors, multiple dimensional arrays.

3.1.2. Keras

Keras is a library that works on top of TensorFlow. The advantage of using it, is the high abstraction it offers. With few lines of code, it is possible to train a state-of-the-art Deep Learning algorithm. However, due to its own abstraction, it has low flexibility which makes it unsuitable for creating new architectures or making low-level modifications to existing ones.

3.1.3. PyTorch

It is an alternative to Tensorflow, developed by Facebook´s AI Research lab. Pytorch is a branch of the original Torch framework, with the goal in mind of providing Torch with Python support. The basis of the library is the processing of the data using Torch. However, CUDA is also supported as a backend.

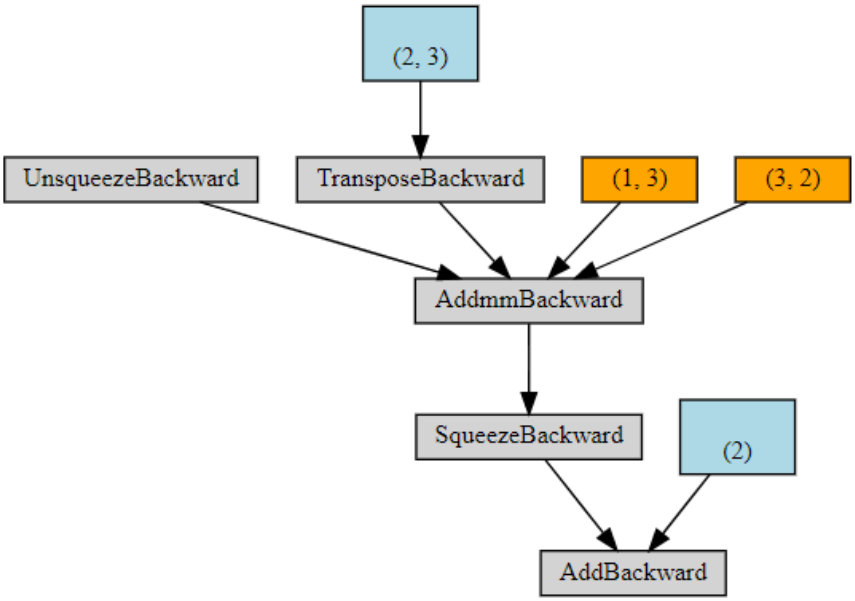

Figure 3.1: A computation graph refers to a graph that represents the operations and how the operations and variables interact between them.

Moreover, the most remarkable feature of Pytorch is the use of dynamic computational graphs. The computational graphs are the execution of the operations represented by the use of graphs. When such graphs are fixed like Tensorflow, the execution of the code is predefined. However, when such graphs are dynamic, the result is defined by a run with the creation of the computational graph. The advantage this feature provides is the possibility of modifying the network on the fly.

Furthermore, the flexibility of Pytorch is high due to the lower-level API focused on direct work with array expressions. This feature made Pytorch gain immense interest becoming the preferred solution for academic research, and applicable to specific applications where high optimization is required. However, with the launch of TensorFlow 2.0 and its support for eager execution, those differences between both frameworks were blurred.

3.1.4. CUDA

The Compute Unified Device Architecture (CUDA), developed by NVIDIA to facilitate the use of their GPUs, is a parallel computing platform and programming model which allows using a GPU for general purpose computing. From a software point of view, CUDA is a heterogeneous programming model in which both the CPU (host) and GPU (device) are used. The use of GPUs is a key factor to train deep learning models.

3.2. Software

The use of libraries makes possible the training and application of Neural Networks in an easy way. However, we still need to develop code in such frameworks. Moreover, we also need data to feed such networks. A big data generation pipeline has been developed for this finality. For the whole pipeline, we will make use of several software resources: Python, Qt, Axis Neuron, MakeHuman, Docker, and CMake. We will proceed to briefly describe each one of them.

3.2.1. Python

Python is an interpreted programming language which emphasizes code readability with indentation and removal of redundant information. It supports multiple programming paradigms. Moreover, Python has acquired a lot of interest due to Machine Learning popularity, since it has become the main language used for this purpose. Its ease of use for mathematicians and scientists is the main selling point for becoming the de facto language for artificial intelligence.

May 2020	May 2019	Change	Programming Language	Ratings	Change
1	2	^	C	17.07%	+2.82%
2	1	v	Java	16.28%	+0.28%
3	4	^	Python	9.12%	+1.29%
4	3	v	C++	6.13%	-1.97%
5	6	^	C#	4.29%	+0.30%
6	5	v	Visual Basic	4.18%	-1.01%
7	7		JavaScript	2.68%	-0.01%
8	9	^	PHP	2.49%	-0.00%
9	8	v	SQL	2.09%	-0.47%
10	21	^^	R	1.85%	+0.90%

Figure 3.2: Ranking of programming languages by TIOBE.

In comparison with other programming languages, the execution speed of Python is not remarkable. However, it can be made quite efficient by the use of wrappers that allow Python to include code in another programming languages such as C or C++.

3.2.2. Qt

The development of our own action data tagger relies on the use of a framework that makes possible the creation of a user-friendly interface. Qt is a modern cross-platform application development platform. We decide to use Qt, instead of other popular platforms like Visual Studio .NET, because it is open source and C++ written, taking advantage of C++ libraries.

The way of developing on Qt is via the use of widgets. The framework itself offers the main widgets we might need. However, it is possible to add new ones by the use of inheritance making it totally customizable. Moreover, a feature that distinct it from others frameworks is the way of interacting the objects. The notifying of changes between objects is made by the use of signals and slots. The signal refers the threshold that is emitted and the slot is

the function that is executed when that threshold is received.

3.2.3. Axis Neuron

The data input to the virtual environment is a BVH animation file. The use of a motion capture suit is needed in order to make the recording. That is why we use Axis Neuron, the associated software to the hardware suit. It allows an easy management of the motion capture system and a fast and easy calibration of the suit. The data can be easily streamed to another environment with the possibility of using the tool as a server. The movement of the suit can be streamed in real time. Furthermore, the recording is easy and fast to do with many configurable options that gives a high level of flexibility (Figure 3.3).

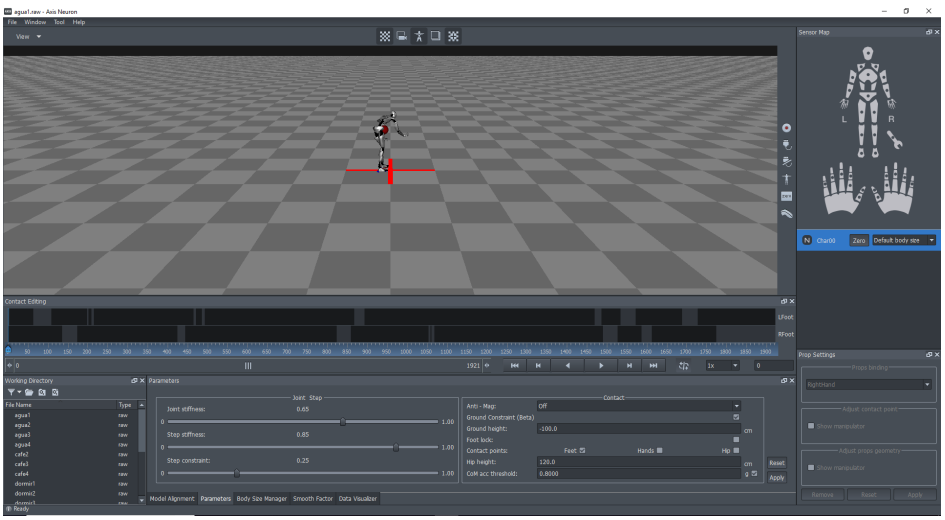

Figure 3.3: This snippet shows the main window of the axis neuron software. Intuitively it shows if the sensors are working properly. Moreover, the animations recorded are accessible. The playing and manipulation of the files with its respective configurations is simple.

3.2.4. MakeHuman

MakeHuman [1] is 3D graphics tool used for the prototyping of photorealistic humanoids models. It is developed by a community of developers, artists and academics interested in the 3D modeling of humanoids.

Its main feature is the possibility of highly personalizing the humanoid (Figure 3.4). Everything in the humanoid can be changed: sex, age, weight and racial origin. Moreover, specific modifications can be applied such as eye color, shoulders forward inclination, face shape, fingers, arms, among others.

An interesting feature is adding a skeleton to the humanoid, totally adapted to the anatomy. Also, it is possible to change to different types of skeletons in order to adapt it to the specific humanoid. Furthermore, the default pose can be changed to some predefined by the MakeHuman tool.

[1] `http://www.makehumancommunity.org/`

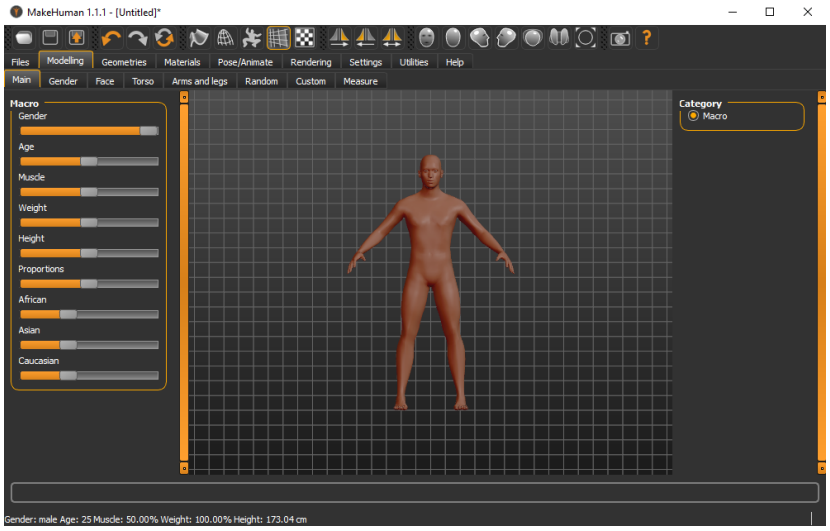

Figure 3.4: MakeHuman tool with high configuration possibilities

3.2.5. Docker

When working on deep learning in a collaborative environment, it is necessary the abstraction of the libraries installed in the computer. This is done by the OS. However, when working with various libraries, but in different versions it is important to make another level of abstraction. Docker accomplishes this abstraction by the creation of containers. Like another OS virtualizers, it isolates applications. However, the isolation is made at software level, not hardware level as VirtualBox. This means, Docker creates a direct connection with the hardware and does not virtualize it's own (Figure 3.5).

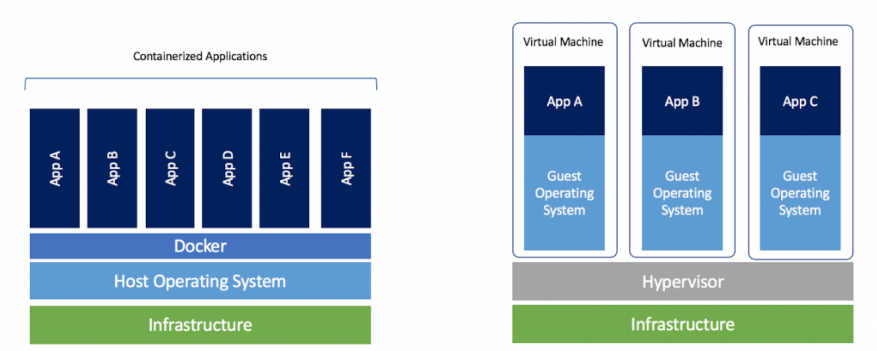

Figure 3.5: Comparing containers and virtual machines. The main difference is the level of virtualization. On the left side, the containers just virtualizes applications and maintains the same hardware and OS. On the other side, a full OS is virtualized. Extracted from `https://www.docker.com/`

Docker works by the use of Dockerfiles: files where the dependencies are described. Then, this file can be built and, as result, an image is obtained. These images are executable and

have all the necessary software installed. Furthermore, it makes possible the extension of another dockers. The result is a fast tool of virtualization and reproducibility.

Docker is the fastest and easiest way of working in a collaborative environment where each one is developing it's own projects with the same share resources.

3.2.6. ROS

ROS is a framework used for working with robots transparently. This means it has compatibility with virtual and real world, being ROS an abstraction layer and working interchangeably. It offers an easy management of the framework, giving fasts ways to interact with the robot and getting the interaction with a distributed system.

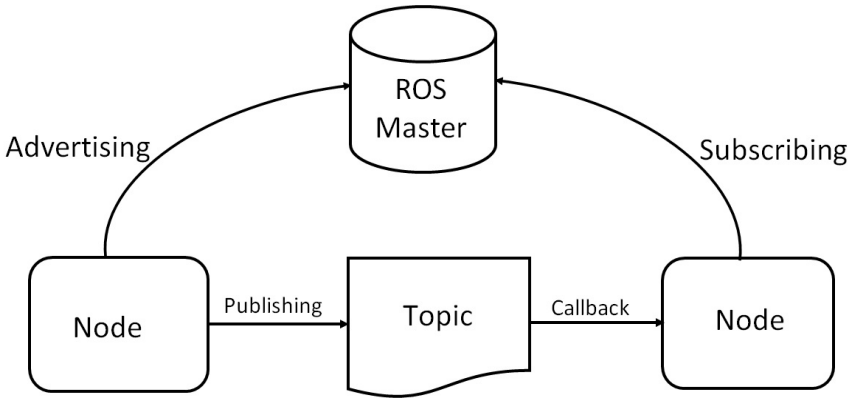

Figure 3.6: ROS workflow

The way it works is based on the use of messaging (Figure 3.6). Each computation process is called a Node. They can publish and subscribe to the topics, where the information flows through.

This simple mechanism makes possible develop software in a Distributed System with real robots, or virtually simulating it.

3.2.7. CMake

CMake is an open-source, cross platform tool that is used to control the software compilation process. It makes possible the generation of native makefiles and workspaces that can be used in the compiler environment of choice. This is totally useful if a project does need to be configured and generated for another compiler or environment.

On the building process, CMake generates the main building files from the configuration files. Then, the native tools of the platform are used for the real building.

The file that helds the building information are the *CMakeLists.txt*. In each file, some commands are defined with arguments determining how to build the code and where to install it.

3.3. Hardware

My research team has offered to me the access to two powerful machines. It is known, that DL needs a lot of computation power in order to obtain results. Therefore, the team provides GPU power and enough storage to accomplish the thesis (Figure 3.7).

Asimov	
Motherboard	Asus X99-A
	Intel X99 Chipset
	4× PCIe 3.0/2.0 × 16(×16, ×16/ × 16, ×16/ × 16/ × 8)
CPU	Intel(R) Core(TM) i7-5820K CPU @ 3.30GHz
	3.3 GHz (3.6 GHz Turbo Boost)
	6 cores (12 threads)
	140 W TDP
GPU (visualization)	NVIDIA GeForce GT730
	96 CUDA cores
	1024 MiB of DDR3 Video Memory
	PCIe 2.0
	49 W TDP
GPU (deep learning)	NVIDIA GeForce Titan X
	3072 CUDA cores
	12 GiB of GDDR5 Video Memory
	PCIe 3.0
	250 W TDP
GPU (compute)	NVIDIA Tesla K40c
	2880 CUDA cores
	12 GiB of GDDR5 Video Memory
	PCIe 3.0
	235 W TDP
RAM	4 × 8 GiB Kingston Hyper X DDR4 2666 MHz CL13
Storage (Data)	(RAID 1) Seagate Barracuda 7200rpm 3TiB SATA III HDD
Storage (OS)	Samsung 850 EVO 500GiB SATA III SSD

Figure 3.7: Asimov specifications

In order to make possible the data collection, some input is needed. In this case, it is the use of visual and motion data. One important input is the use of a motion capture suit, which can track the movements where wearing it. This is helpful for virtual data. The suit used is the Perception Neuron Version 2.0. It is a non-optical motion capture type using Inertial Measurement Units. Specifically, each location to be tracked in the body requires a Neuron. Each one is composed by a 3-axis gyroscope, 3-axis accelerometer and a 3-axis magnetometer. These 3 are used due each its own advantages which by combining them allows for quick and accurate position and orientation determination with a low drift over time. For example, a magnetometer has poor accuracy for fast movements, but almost zero drift over time. However, when using gyroscopes the reaction of movement is quickly and accurate but it accumulates a lot of error over time. It supports a total of 32 neurons distributed through the whole body. The noticeable is the possibility of full hands tracking with 7 neurons on each hand, including the rest of moving parts of the body.

Another capture input data is the use of cameras. The result are videos with RGB format. Moreover, with the use of pairs of stereo cameras, videos with depth can be also obtained. These two use cases are reached with the use of 2 kinects and 1 PrimeSense. PrimeSense is the precursor of the famous kinect. They can be easily interconnected due to compatibility.

4. Development and Experiments

This third chapter describes more deeply the development of the project, that is, how all the software and hardware is combined. Section 4.1 describes specifically the data generation pipeline. Section 4.2 specifies the pipeline used for learning and using the data obtained.

4.1. Behaviour Data Creation Pipeline

This section presents the data creation pipeline. It specifies the work done in this field starting from the behavioural sequences recording. This data can be feed to the UnrealRox tool presented by the 3DPerceptionLab Team. We present an extension that makes possible a behaviour analysis data generation. As a result, it outputs sequential data generated which can be passed by a Tagging Tool. In this case, a tagging tool work is presented for making possible this task. Finally, we present a video data augmentation algorithm for action classification.

4.1.1. Scene Recording

Scene recording is an important step previous to the data generation. We propose a recording method which makes possible to create own data for behavioral purposes. From this recordings, data is generated in different formats: RGB, Depth and BVH animations.

In order to properly record, a previous fully configurable setup is needed which is not straightforward. The required hardware consists on: 4 computers, 2 kinects, 1 primesense and a motion capture suit (Figure 4.1).

On the other hand, the software also must be configured. One problem that appeared with the preparation of the software is the use of different OS. This resulted in compatibility issues between the computers which hindered the setup. For example, when working with ROS, all computers are required to have the same version installed. Moreover, in order to satisfy this last condition, the same OS must also be installed. In order to overcome these problems, Docker has been used successfully.

However, the use of Docker is not trivial. In order to create a valid Docker image, the Dockerfile must contain all the needed steps to install the complete OS to suit the recording process requirements. In addition, it requires the understanding of the interaction with the main Operating System. On one side, Docker needs access to the USB ports for receiving the information from the Kinect sensors. Moreover, the persistence of the data must be configured in order to maintain any change made inside the Docker container (Figure 4.2).

Once the software and hardware are configured, the deployment of the setup is possible. In order to start the docker on each computer, ssh is used as each IP is known and static. Furthermore, the distributed ROS system is prepared, meaning that each computer which participates in the ROS network is configured. Specifically, the next steps are followed:

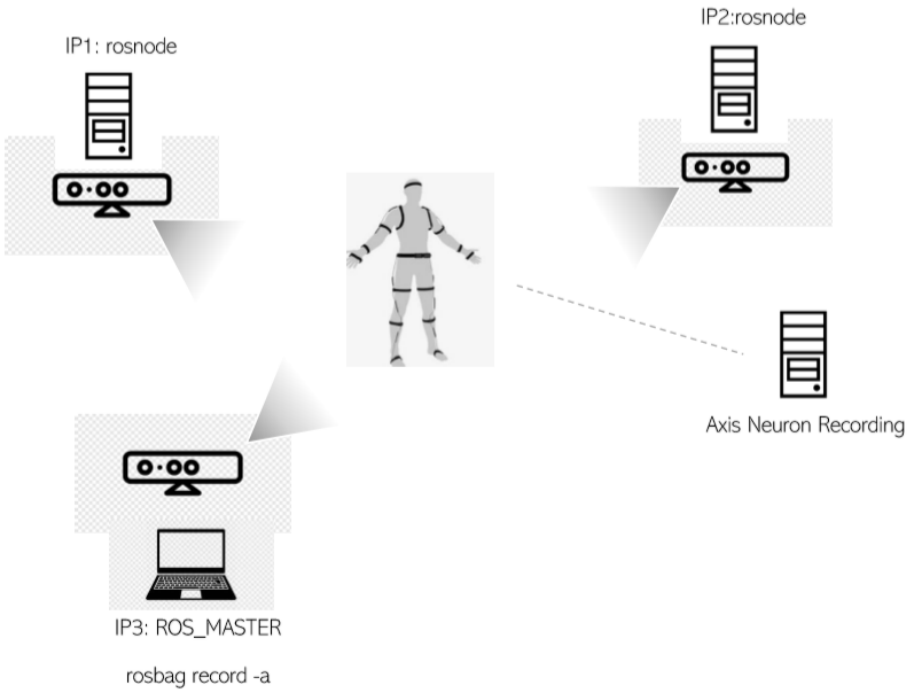

Figure 4.1: Setup of the recording of behavioral data

1. Start roscore in the main computer, referred as the master.
2. Check if computers are accessible with ssh.
3. On each computer called as slave define the ROS_MASTER_URI=http://IP:PORT.
4. Test if it works by checking that the topics are visible to all.

At this step, we will focus on the motion capture suit from Axis Neuron. We are using the full body configuration, i.e. full body tracking. When put on, it is required to follow a process of calibration with the use of its software. This helps to assure that the sensors are laying in the correct position to accurately capture the motion data.

Once all is correctly deployed, it is easy to start the recording. In the first place, in order to start the recording with Kinect cameras, the rosbags are used. We defined them to record all the topics that appeared visible for flexibility purposes. The command used is *rosbag record* with the parameters "-o" to give it a name and "-a" to record the data of all topics. If we want to reproduce the data, we can use the *rosbag play* option. Then, by using the "–hz=" parameter it is possible to process the data slowly without needing to deal with real time speeds.

For each behaviour, a new rosbag and a new BVH file is created. A total of 5 behaviours are recorded: preparing a glass of water, preparing a cup of coffee, sleeping, opening a door to a friend and tying shoes. Each one is recorded 3 times for variability purposes. Some variations added are: the order of the actions, the position of the subject, the positions of the cameras...etc. The goal is to obtain the most realistic and diverse data possible.

```
export containerName=ros_kinetic

sleep 3 && \
        xhost +local:`docker inspect --format='{{ .Config.Hostname }}' $containerName` >/dev/null 2>&1 &

XSOCK=/tmp/.X11-unix
nvidia-docker run --rm -it \
        --volume=/home/john/daniel/Synthetic-Pipeline:/mnt/workspace/ros-sync/Synthetic-Pipeline:rw \
        --volume=$HOME/.Xauthority:/root/.Xauthority:ro \
        --volume=$XSOCK:$XSOCK:rw \
        --volume=/dev/pcan32:/dev/pcan32 \
        --volume=/dev/pcan33:/dev/pcan33 \
        --volume=/dev/pcanusb32:/dev/pcanusb32:rw \
        --volume=/dev/pcanusb33:/dev/pcanusb33:rw \
        --volume=/dev/pcan-usb/:/dev/pcan-usb \
        --volume=/dev/pcan-usb/0/can0:/dev/pcan-usb/0/can0 \
        --volume=/dev/pcan-usb/1/can0:/dev/pcan-usb/1/can0 \
        --volume=/lib/modules:/lib/modules:ro \
        --privileged \
        --env="DISPLAY" \
        --env="QT_X11_NO_MITSHM=1" \
        --net=host \
        --name $containerName \
        kinect/ros:latest bash -c "source /mnt/workspace/ros-sync/Synthetic-Pipeline/sync-project/devel/setup.bash && export ROS_IP=172.16.
```

Figure 4.2: Script for executing the docker file with USB port forwarding and persistence storage. Moreover, it executes the ROS setup and initiates the OPENNI driver for reading the kinect data.

The first behaviour of preparing a glass of water is simple. The actions that are used to perform the behaviour in order are: walk towards water and glass, move glass, take the water container and spill the water in the glass and, finally, take the glass and leave.

One more complex behaviour, which is formed up by more actions than the rest, is the preparing a coffee cup behaviour. Firstly, as the rest of behaviours, the person walks towards the place for preparing the coffee cup. Then, it opens the coffee machine and inserts a coffee capsule. After closing the coffee machine, the cup is put on. Finally, when the machine finished, the cup is taken out. However, as simple the behaviour seems, a lot of steps are required to prepare a coffee cup. Moreover, these steps can be followed in different order. For example, the first step before opening the coffee machine can be the action of placing the coffee cup in the machine. Furthermore, sometimes actions can be omitted such as inserting a coffee capsule, which can happen when in another moment a new one has been already inserted.

Another interesting behaviour is opening the door to a colleague. The interest relies in the possibility of recognizing that the door has not been closed yet. This will prevent to happen misfortunes as robberies. The full behaviour is formed up by 4 actions. Firstly, when a colleague calls to the door, the person walks to the door. Then, the person opens the door and receives the guest by greeting him. Finally, the last action is the closing of the door which is an important step.

The behaviour of sleeping is of special interest (Figure 4.3). It is useful to learn to detect when a subject falls asleep, waking him up and advising him to find a better place for continuing to rest. The behaviour is made up by 4 sub-actions: walking, sitting down, working and sleeping. Specifically, the steps are: walking towards the chair, sitting down,

working on the computer and unexpectedly sleeping.

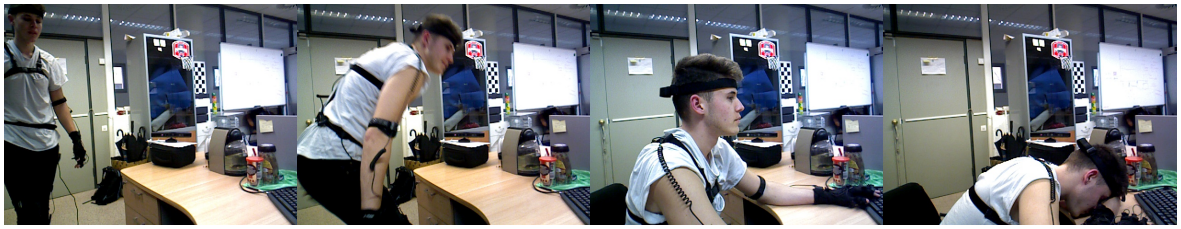

Figure 4.3: Behaviour of sleeping made up by 4 actions. It is possible to visualize another kinect which is recording the same behaviour from a different perspective.

Last but not least important, we have recorded the tying shoes behaviour. Its relevance comes with the goal to record the full actions required to tie the shoes. In this case it has a total of 10 actions: walking to the shoes, picking up the shoes, sitting down, putting on left shoe, putting on right shoe, tying left shoe, tying right shoe and, finally, stand up and keep the walking with the shoes on. As in the rest of behaviours, some steps can be omitted or the order of the actions can be changed.

4.1.2. UnrealRox

UnrealRox is a virtual environment for data generation. Specifically, it has the possibility of recording the movements of the agents and objects in the scene with the use of cameras and output the images captured from them. As result, the framework generates the dataset with multiple formats. Mostly, the main goal is to use that data to feed Machine Learning algorithms. Therefore, the data must be as much realistic as possible, in order to obtain good results. We propose an extension of the framework for making procedural data generation and integration with BVH animations.

When creating the Unreal Engine (UE) project, we take as basis the UnrealRox project and make all the modifications over it. Firstly, the virtual environment must be prepared. In order to obtain photorealistic results, we have integrated the *Unreal Engine Interactive House* [1] on UnrealRox. It is made up by 2 main rooms with a lot of details as furnices and small artistic adornments. Moreover, it supports the physical interaction with some furnices.

Then, once the photorealistic environment is integrated, it is possible to load it on night or day time. In our use case, we have chosen the day time style. The reasons are because once the full system is developed the goal is to deploy it in houses with the elderly. In that case, the system must be mostly active at daytime. Furthermore, it is required to clean some elements such as the main character used.

The way UnrealRox works requires the deployment of ROXcameras. which are specific cameras used by UnrealRox, and a main actor, which gives access the system to the scene. However, the placement of each ROXcamera in the scene is not trivial (Figure 4.4). When considering where to locate the cameras, it is important to take into account for what and where the data will be used. Firstly, whichever case use, the most important thing is the

[1] `https://ue4arch.com/shop/complete-projects/interactive-house/`

perspective variation of the cameras. Then, it is relevant to consider where will the data be used in order to obtain the most similar data possible. In our case, we have used a total of 4 cameras: 3 cameras at the same approximate height with the higher possible perspective variation and the last camera with a different angle and height. The setup of the three first cameras at same height but from different views results is interesting since provide similar perspectives than a possible mobile social robot. Moreover, in one of the cameras views, some occlusion is added which makes the data more adjusted to real world situations. Finally, the last camera contributes a total different view which can be beneficial for more generalized data.

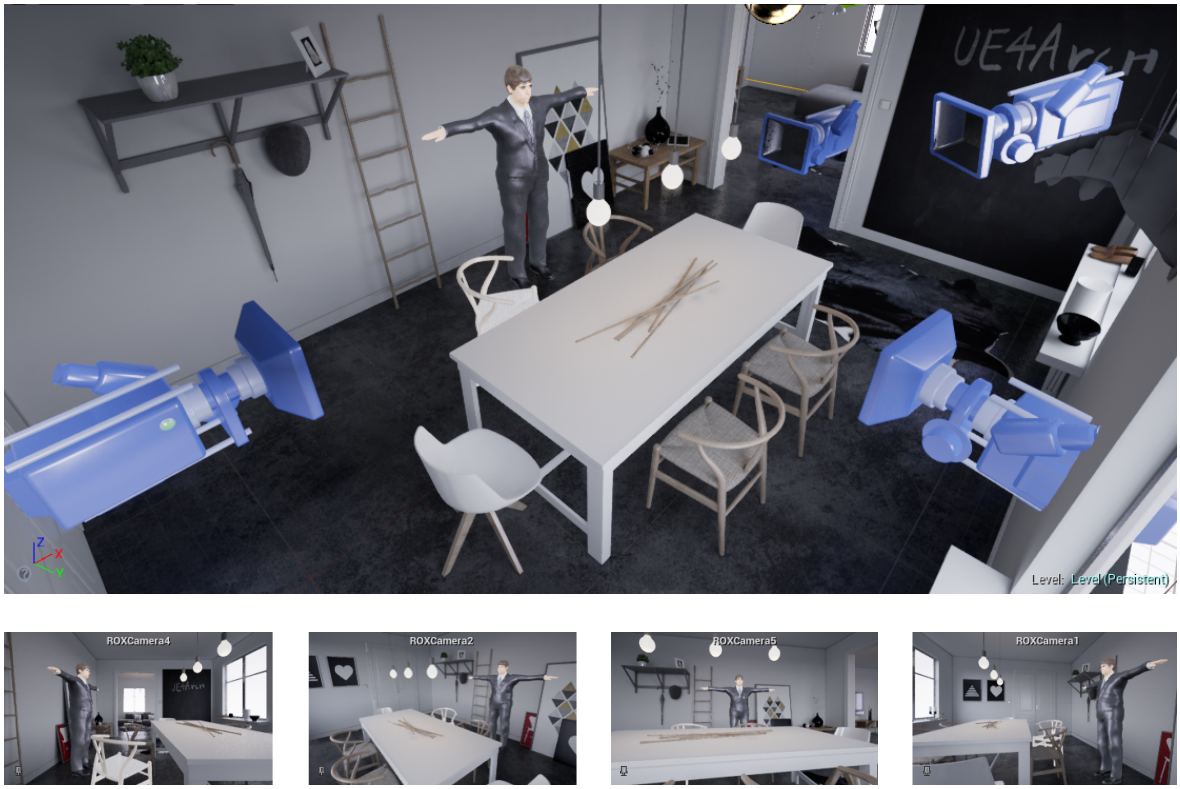

Figure 4.4: Snippet of the placement of the ROXCameras and the outcoming views of each camera.

After preparing the environment, the next step is adding the possibility of importing recorded BVH files and applying it to a skeleton of a character. This process is not trivial and can be problematic. Specifically, when adding a BVH to a character it is important to deal with the differences of both skeletons: the character one and the BVH file one. It is important that each bone of each skeleton associates correctly because each relative bone movement from the BVH file is applied on the character skeleton. Unreal Engine does not offer a fast solution to make possible this. However, the same creators of the Axis Neuron hardware and software, have created a plugin for UE that facilitates this. In order to install the plugin, we have to download it and extract the files inside the plugins folder of the project.

Moreover, diverse rigged characters must be used in order to contribute enough diversity

for the resulting synthetic data. The open source MakeHuman tool is used due to being free to use, its high flexibility and speed of rigged humanoid models creation (Figure 4.5). It returns the corresponding humanoid in a importable FBX file.

Figure 4.5: Rigged humanoid models used as meshes for the pipeline

Once imported the rigged meshes, the retargeting process can be started. All the bones are associated to a global bone system provided by the Axis Neuron plugin. In order to accomplish this, some steps are followed. Firstly, when imported the model, many files are created: the mesh, the bounding boxes, the skeleton and many secondary files. From the skeleton asset, we create an animation blueprint. When opened, we add a new node *NewPoseCalc* which comes with the plugin. Then, it is possible to start the retargeting process, assigning each bone from the skeleton of the mesh to the bone from the main axis neuron skeleton (Figure 4.6).

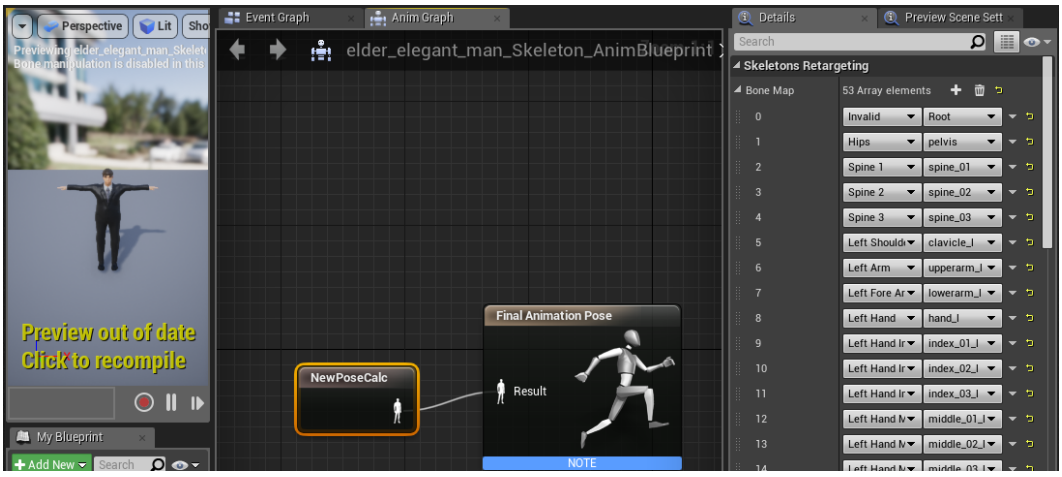

Figure 4.6: The resulting animation blueprint from the mesh which with a import node *NewPoseCalc* makes possible the retargeting of the bones. In the Skeletons Retargeting zone the left side is main skeleton system used and in the right side are the specific bones of the skeleton from the mesh.

The next step is to create a new actor which is formed up by the retargeted skeleton and extra components which helps to manage the animation. We create a new Blueprint class

whose parent is an actor. It is required to add two components: the first is the *PerceptionNeuron* component which gives the option to add a BVH file or a live transmission from the Axis Neuron software. The second component is the *SkeletalMesh* which adds the mesh and the animation blueprint created (Figure 4.7).

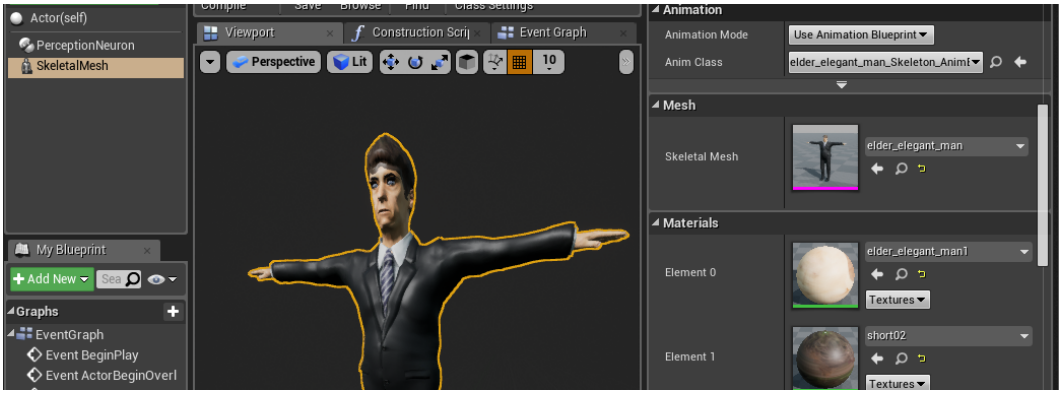

Figure 4.7: The resulting configuration of the Blueprint actor with the *PerceptionNeuron* component and the *SkeletalMesh* configured.

Finally, it is possible to add the actor to the scene and assign it a imported BVH asset generated from the AxisNeuron software (Figure 4.8).

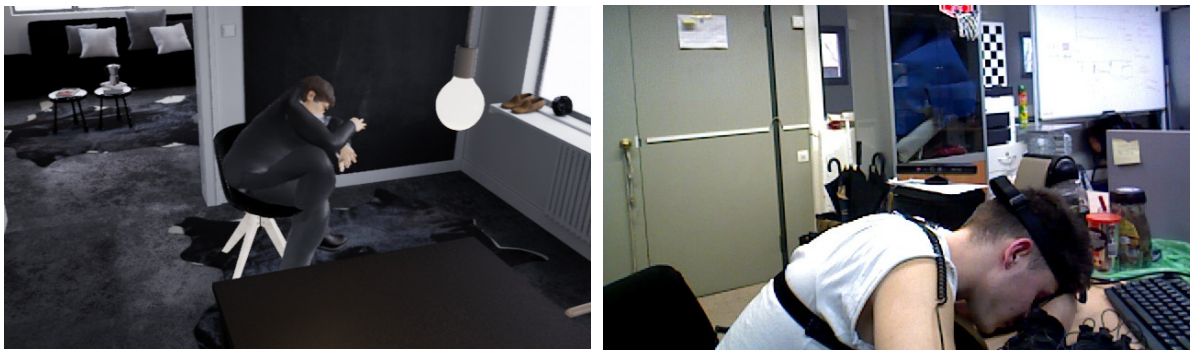

Figure 4.8: Comparison of real data with synthetic data results. This is made possible by recording simultaneosly motion capture data and real data with the kinects. View A.2 for more samples

4.1.3. Action Manual Tagger

In every Deep Learning task, data is one of the most important resources for obtaining properly good results. This data can be tagged automatically by using virtual environments or manually by humans. Each one has its own advantages and disadvantages. The tagging in this case is an video oriented one. Specifically, the result are videos tagged with which action are made in each frame.

To automatically tag data, we needed to create a virtual environment where each action and state must be perfectly known in order to create the dataset. Moreover, the virtual environment should make possible a good enough simulation in order to obtain good data. This process is the most tedious one. Most of the time is dedicated to the development of the software. However, once finished, the data that can be created is immense. Also, the problem is that the resulting data is artificial, which is harder to transfer to real world problems.

In the other case, the manual process is faster. The time consuming part is the recording and tagging itself of that data, not the development of the software. However, the task of obtaining the data is slower, depending on human operators. In this case, when a lot of data is needed, this is not as scalable as the first option.

In this context, we propose our own manual data tagger for videos of actions. This tool is fully useful for making possible the manual tagging of videos and images in all possible formats. Moreover, a more specific format is the BVH. This format is useful as the main problem is the action recognition in humans. The behaviour can be summarized in just the basic skeleton of a person.

The behaviour of a human can be defined at different levels. The highest level can be the behaviour that the person itself is carrying out. At the lowest level, it can be defined as specific as the movement of a single finger.

There are a maximum of three action tagging levels, meaning that the tool can create a hierarchy of actions of height 3: behavior, actions and subactions. The behavior is composed by actions, and for accomplishing those actions, many subactions are defined. For example, a behavior could be making a tea. All the actions would be: moving the cup from place to place, warm water, filling it...etc. In the case on moving cup from place, resulting subactions are: moving left arm up, closing left hand...etc.

It makes possible to give a frame annotation level, reporting in each frame which behaviour, action and subaction is happening. As result, a behaviour analysis dataset is obtained. This information can be used to train Deep Learning or classical oriented algorithm.

Since tagging in action oriented problems is a tedious process, our program is an user-friendly window with clear instructions and totally adjustable (Figure 4.9). The tool is totally visible in just one window, making the tagging process faster and avoiding the need of changing between windows. Every option is accessible and clearly different from the rest.

In order to start the tagging, loading a file with visual actions information is needed. The supported formats are diverse: BVH, video and images files. Firstly, the type of file is selected via tabs at the top of the player. Then, the tool checks if the input format is supported by the tool itself. Given the type of data, the manual tagger works totally transparent, meaning that the tool does work the same, independently of the file format. This is possible due the use of inheritance, an easy way of quickly adding new formats without modifying code, just adding new classes totally decoupled.

One problem we could identify is related to the file formats use for video visualization, due to the limitations that Qt has. No widget supports a multiple video format input with a frame by frame control. This resulted in the need of an own widget that supports this functionality. This was solved creating a media player with the use of the OpenCV library.

An interesting format already presented is the BVH. The development of a BVH player is complex, due to the need of managing a hierarchy of bones and the consecutive movements frame to frame. In order to solve this issue, an external tool is integrated. This animation

Figure 4.9: Complete interface Action Tagger. It is divided in 3 main components: media player, create new behaviours/actions or subactions and add the new one created as sliders.

BVH player is the opensource BVHplay tool [2]. The tool is developed in Python with the interface library Tkinter. It required the understanding of the code itself in order to make a properly integration.

As the tool is written in Python and our tool in C++, the integration cannot be made directly. Two separately windows are needed in this case. However, this is not problematic because the action tagging interface is totally adjustable giving enough room for two windows in one screen.

The communication between both windows is done via the use of bridges with synchronised sockets. A messaging protocol is defined in order to support all needed instructions via raw messages like: set frame, get frame, get total frame number, play, stop...etc. Moreover, the media control in the BVH animation player must be removed, since the control being delegated to our tagging tool. In combination, it makes possible a live and instant communication between both tools.

In order to use the tool with BVH animation it is needed to start both tools, in any order. Firstly, the BVH can be started with the use of Python2.7. The entry point of the tool is the bvhplay script. This will start a tkinter interface window. Then, with the start of the tagging tool and the BVH tab chosen, it is possible to control the BVH.

When both tools are started, a BVH file can be loaded. From this point, the communication can be effectively started. Each tool giving enough information between them with a request and answer protocol.

[2]BVHplay tool: `https://sites.google.com/a/cgspeed.com/cgspeed/bvhplay`.

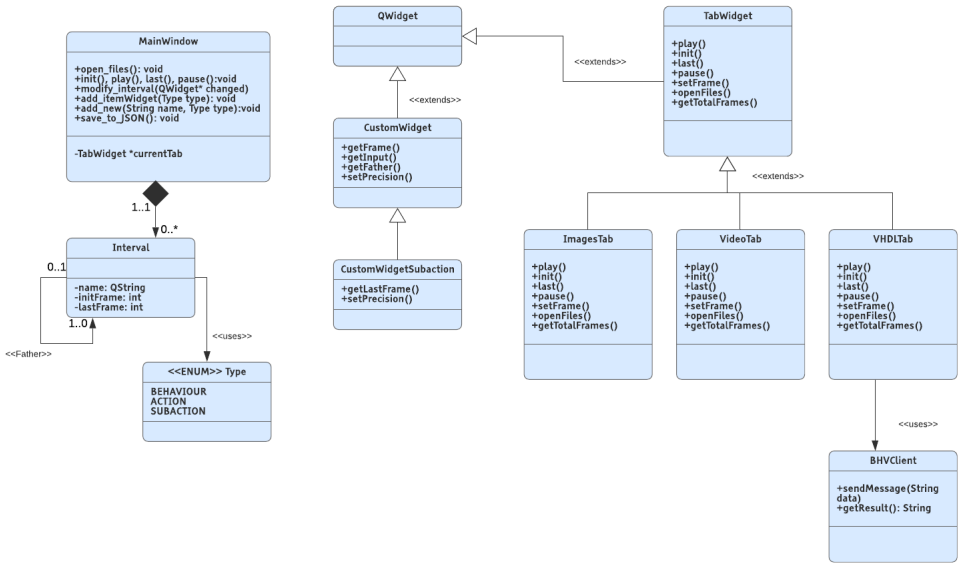

Figure 4.10: The UML of the tool. It represents the main information, ignoring redundant information. Moreover, the communication between the BVH player is not included.

Once added the file to the tool, behaviors, actions and subactions can be manually defined. This is done with the text box and the buttons with the "new" prefix. All new ones are added in three lists referring to each level.

Then, lastly, they can be instantiated into the form of sliders. This is done using the textboxs with the "add" prefix (Figure 4.12). It adds new sliders with a default label to the corresponding level. With a drop-down list the correct label can be chosen. Moreover, the slider can be adapted to define from where to where that behaviour/action or subaction embraces. Each level is visually distinguishable with colours.

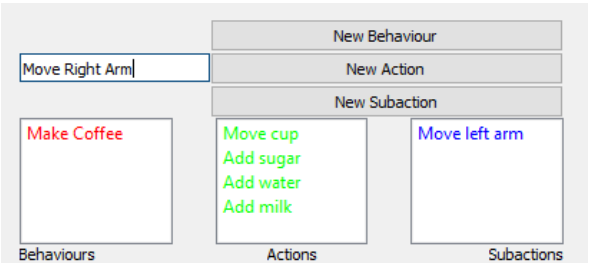

Figure 4.11: Clear options for creating new behaviours/actions and subactions. They can be added to its respective list. Next, sliders can be instantiated with a label assigned.

Two type of sliders are used. The first one which just defines the end of the behaviour or the action. The second which is a compound slider making possible to define the start and end of a subaction. This way, it makes possible to allow overlapping between another subactions. For example, in the behaviour of making tea it has the action of filling water. This action is formed by more subactions like moving right arm to grab the hot water and moving the left

one to grab the glass with the tea bag. These two subactions are done simultaneously.

Moreover, defining the first type of slider this way, it results in behaviours and actions depending by the one before. The advantage is that helps the tagging to be more visual and each frame more identifiable to which behaviour/action or subaction it belongs.

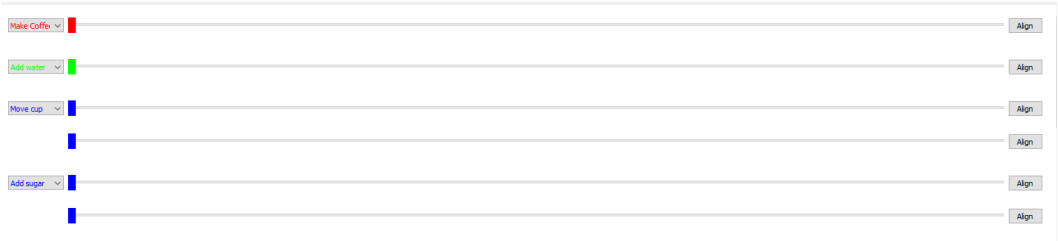

Figure 4.12: Slider examples

A global slider was included in order to play and go through all the frames of the file. This slider is required as the rest are based on the information outcome of the main player. All the sliders are aligned to this global one, so each one can be easily adjusted by a button to the main slider.

In addition, this video player slider is added with main buttons that conforms the full video player controller: pause, play, set to start and set to the end.

The sliders are the visual part of the structural storage of all the tagging. In the backend, the sliders are represented as a linked list where each element saves its parent, the frame range, name and which type of behaviour represent. This helps to decouple the frontend from the backend.

Finally, once the tagging process is ended, the information of the sliders can be dumped in a JavaScript Object Notation (JSON) file. In this case, the JSON used for C++ is by nlohmann[3]. The dumping could have been made with our own format. However, JSON has the advantage of having a high compatibility with different OSs and programming languages. In this case, the most important is to define a good data format of the JSON file when the retrieval of information is necessary.

4.1.4. Video Data Augmentation

When we are dealing with Deep learning, it is interesting to have a big amount of diverse data. If the data lacks of variability, the training process would be difficult to succeed. Data augmentation is one of the best techniques to improve data by incorporating variation. We propose an extension of the Action Manual Tagger which returns augmented data.

For example, with this video data augmentation pipeline, one possible variation is changing image contrast to make the DL model invariant to light changes. Furthermore, the use of shifts, rotations and cropping can help the data to be more spatially varied. Applying this operations, the neural network trained with this data learns not to focus to fixed locations of an image.

The video data augmentation is an extension of applying image data augmentation over the same frames of a full video. Each operation processed over the first frame is also applied

[3]JSON library for c++ by nlohmann: `https://github.com/nlohmann/json`.

to the rest of the video. Therefore, the typical operations over images can be used on videos as well (Figure 4.13): rotation, translation, vertical and horizontal flip, cropping, changes in brightness, add noise or blur, among others.

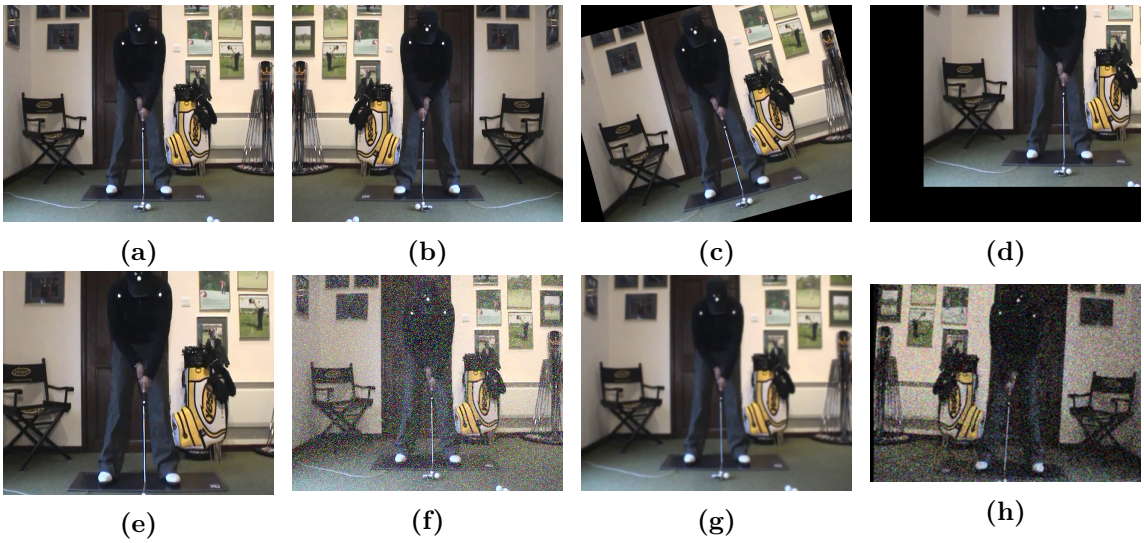

Figure 4.13: Figure representing own data augmentation operations applied to the raw image 4.13a: 4.13b applying horizontal flip, 4.13c with random rotation, 4.13d performing a translation, 4.13e featuring a center cropping operation, 4.13f introducing gaussian noise and blur in . Finally, an example of applying a random list of previously specified data augmentation techniques 4.13h.

Moreover, when dealing with videos there are other possible data augmentation techniques. For example, variations regarding the temporal dimension changing the frame rate by adding or removing frames randomly. When dealing with behaviour analysis, this improves model stability as there is a variation in the speed of the action execution.

In detail, it is capable of reading videos (in popular image/video formats) and the JSONs file associated (Figure 4.14). The input structure required is to separate the videos and JSON files in folders. Each video, is splitted in individual frames, that are also divided in folders. The JSON files names and each video folder names must be the same to be recognizable.

In order to process the videos correctly, the videos must be divided at action level. This means that one video results in multiple videos, each one describing a specific action of the full video. Specifically, the JSON files are read and the videos are separated, a new video results from all the consecutive frames with the same label. Then, the videos can be augmented with configurable parameters. Firstly, scale parameter represents how much varied videos are extracted from a same video. Then, there is a parameter which limits the probability of applying one operation over the video. There are other configurable parameters such as: rotation variation, translation range, crop pixels, duplication of frame probability, range of frames duplicated, brightness variation...etc.

Finally, once the data is preprocessed and augmented, it is exported in order to be usable. The processed videos are converted to real videos using the OpenCV library. Moreover, two different file texts are generated. The first one indicates the frame-label correspondence.

```
{
  "actions": [
      "move_left_foot",
      "move_right_foot"
  ],
  "all": [
      "move_left_foot",
      "move_right_foot",
      "walking"
  ],
  "behaviours": [
      "walking"
  ],
  "frames": 15,
  "name": "walking_cat",
  "sequence": {
      "actions": [
          1,1,1,0,0,1,1,0,0,1,1,1,-1,-1,-1
      ],
```


Figure 4.14: Example of a JSON file describing the behaviour of a cat walking. In all the 15 frames, each frame has assigned which the action is occurring at that moment. The numbers refers to which label in the array "all" it corresponds. Moreover, "-1" indicates that non action is produced in that instant

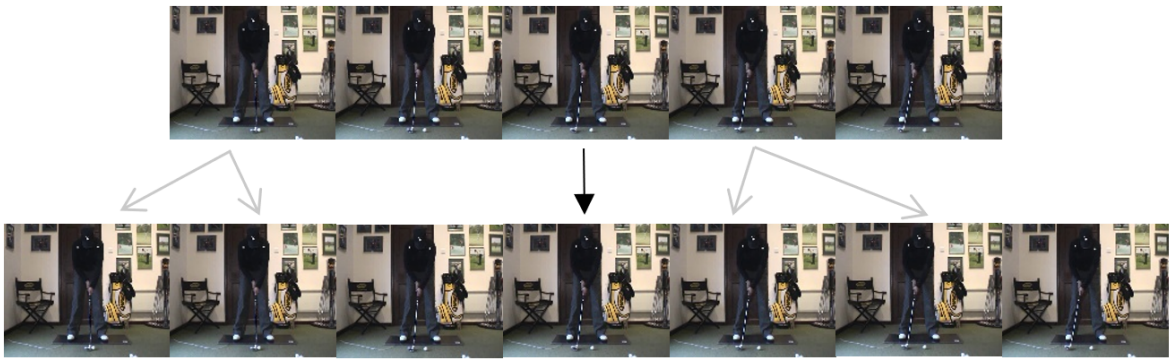

Figure 4.15: Example of a video data augmentation operation by adding frames. It is a person doing the action of swinging back the golf club.

And the second one refers to the dataset file text, i.e., it assigns each video the corresponding number label.

4.2. Deep Learning TSN

Firstly, an in-depth analysis of the Temporal Segment Network is needed. The Temporal Segment Network [24] is one of the best existing approach for large context action classification. It combines a sparse temporal sampling strategy and video-level supervision that enables efficient and effective learning using the whole video. Its potential relies in the way of extracting relevant information.

In image classification, CNNs have witnessed great success. Moreover, they also have been introduced to solve the problem of video-based action recognition. However, the application of CNNs is impeded by two major obstacles. First, long-range temporal structure is important

in the understanding of the action videos. The CNN usually focus in short-term motions, not capable of learning long-range information. This problems are tackled by using fixed sampling intervals. But, this approach requires excessive computational cost when applied over long video sequences. Second, the CNN requires a large volume of training samples to obtain optimal performance. The problem is that the available data remains limited, resulting in a very deep CNN with high risk of over-fitting.

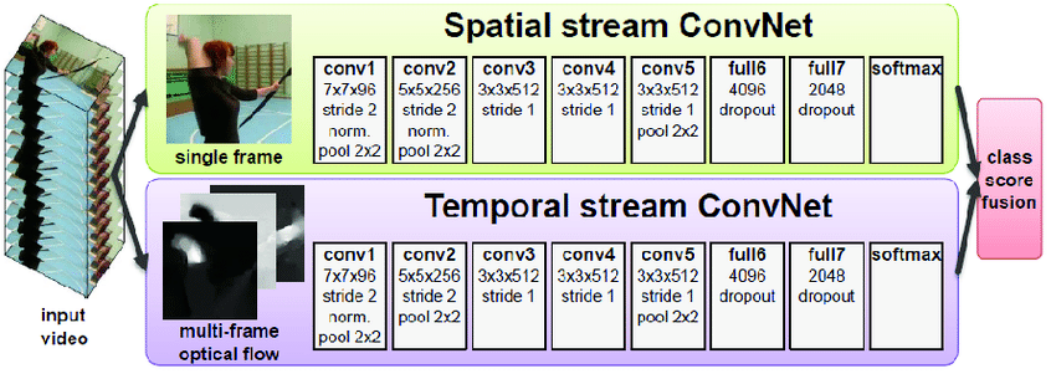

Figure 4.16: Two stream network used as building block by TSN. Figure extracted from: `https://www.researchgate.net/figure/Two-stream-Convolutional-Neural-Network-CNN-architecture-Source-148_fig3_312642887`

The TSN works on top of the successful two-stream architecture represented in Figure 4.16 and consisting of a temporal and a spatial stream. A key observation in the temporal structure is that consecutive frames are highly redundant. Therefore, dense temporal sampling is not needed. Instead a sparse temporal sampling strategy will be more favorable. Applying this observations, results in the new video-level framework Temporal Segment Network. In summary, it extracts short snippets over a long sequence uniformly along the temporal dimension. All the information is aggregated using a segmental structure.

The technique to accomplish this works by dividing the whole video V in K parts with same duration, resulting in $S_1, S2, ..., Sk$ segments. Then, a small snippet T_k is randomly obtained from each part. Each segment S_k has its snippet T_k associated. This snippet conforms a tuple of two type of values: spatial information and temporal information.

Each snippet T_k is passed to the function $F(T_k; W)$ where F represents a CNN with parameters W which operates over the snippet T_k. The result are class scores for all the classes. Once applied this function to each snippet, the segmental consensus function G combines the outputs to obtain a consensus of the classes among them. Finally, H predicts the probability of each action class for the whole video.

The class score G_i is inferred from the scores of the same class on all the snippets, using an aggregation function g that can denote an averaging, maximum, weighted averaging operation, among others.

Training the temporal segment network is sumarized in the following steps: (1) cross-modality initialization, where pre-trained weights are used for both CNNs as initialization with simple transformations; (2) regularization techniques as partial batch normalization,

where the mean and variance parameters of all Batch Normalization layers are frozen except the first one. Moreover, a high dropout is applied after the global pooling layer in order to reduce more the effect of over-fitting; (3) data augmentation operations, like horizontal flipping and random cropping, which helps to generate diverse training samples and helps to prevent over-fitting. Moreover, two new ones are applied: corner cropping, where only the center or the corners regions are extracted from the image. The second operation is an modified scale jittering. The width and height of cropped regions are randomly selected between four fixed values: $256, 224, 192, 168$. Then, these extracted regions are resized to $224x224$. It involves scale jittering, but also aspect ratio jittering.

4.2.1. Experiments

We have deployed a pre-trained TSN on the UCF-101 database. Specifically, we experimented the neural network with self recorded data. Specifically, we have checked the results using the "preparing a coffee cup" sequence.

Firstly, in order to setup the TSN, we have loaded the network on PyTorch[4] within a Docker. It is required to load the trained weights on the UCF-101. This will prevent the long process of training the TSN by ourselves. Moreover, it exists the possibility of fine tune the network and adapt it to a new dataset in much less time. In our case, we have used the default configuration with no retraining. It uses the RGB input format with no optical flows. Moreover, the base model used is the BNInception with a dropout ratio of 0.8.

Figure 4.17: Example frames extracted from the sequence "preparing a cup of coffee" passed through the TSN. It is the output of one kinect camera.

Once the TSN is prepared, the input data must be preprocessed in order to be passed through the network. The sequence "preparing a coffee cup" is extracted completely from the rosbag and converted to a full mp4 video (Figure 4.17). We have chosen this sequence due its complexity and all the actions occurring in it. Moreover, the TSN requires the creation of a file text which contains the next information: the videos files path locations, the frame snippets and the class label assigned. In our case we are only testing the prediction of the network, being the assigned class irrelevant for our experimentation.

Then, after following these steps, it is possible to execute the neuronal network and obtain the prediction. As already mentioned, the TSN extracts randomly snippets from the input clips which are passed through the network in order to return the prediction. When following this process, the TSN classifies the own recorded clips as the "band marching" class. Finally, we have studied why it resulted in that prediction. It is possible that the visible movements

[4]The TSN in PyTorch version used is the Yixiong´s one: `https://github.com/yjxiong/tsn-pytorch`

and the appearance of another coworker in the scene might be confused by a march. Moreover, the result is not consistent due to the small correspondence of the actions recorded by us with the actions on the UCF-101 dataset. This are the two possible reasons that explains this result.

Figure 4.18: Examples of frames extracted from the videos used for training the class "band marching". In this videos it is possible to perceive a lot of fast and coordinated movements from a group of persons.

The initial goal was training the TSN with own generated data: the synthetic data and the self recorded one. However, due to the time constraint, insufficient hardware for generating the data and training on it and, finally, the difficulty to accessing them due the current COVID-19 situation, the goal has not been possible to accomplish.

5. Conclusions

This chapter summarizes the conclusions drawn from this work. It is divided in two sections. First, in Section 5.1 we sump up the work done in this Thesis. Then, in Section 5.2 possible improvements as well as future lines of research are explored.

5.1. Conclusions

In this Thesis, we have started with an extensive review of datasets, data generators and machine learning architectures which have been used in order to solve the problem of action recognition. We have compared the advantages and disadvantages of each method.

To review existing datasets and data generators, we focused on their advantages and disadvantages as well as on their main contribution to the literature. While most of them are of general-purpose, some other are completely focused on the behavior analysis problem. As of general-purpose, the most relevant are UnrealRox, Gibson and Matterpor3D. Focused on behavior analysis, the important ones are Virtual Home, AI Habitat, HMDB51 and UCF101. Closely related to our work we found the PHAV generator[19].

Moreover, we have studied the majority of architectures used in the behaviour analysis field. As result, we have defined three main groups of algorithms: handcrafted, single stream and multiple stream algorithms. In each group, there exists outstanding methods. However, the state of the art in behaviour analysis are the Temporal Segment Networks, followed by the Inflated 3D ConvNets.

In addition, we presented a data generation pipeline featuring a video action tagging tool as main contribution. Its main function is to define per-frame action. It has the possibility of a three level tagging. Moreover, it is compatible with a wide data type formats; for example, it supports the tagging of BVH data being relevant in the human behaviour analysis. On the other hand, some work on UnrealRox has been done. We have developed an Unreal Engine project totally prepared in order to generate photorealistic synthetic data from BVH animations files.

Furthermore, we presented an easy setup for a complete recording of human interaction scenarios. The purpose is to generate high-quality data to train human behaviour analysis models. From the wide range of the generated data, the most interesting ones are the mocap data, depth and RGB data captured from 3 different perspectives.

Another contribution, also taking part in the data generation pipeline, is the video data augmentation. It´s an extension of the video action tagging tool, receiving the data and processing it. As result, a series of diverse videos are obtained from the same data. What´s relevant is the large number of operations that can be applied. Operations at image level as translation, cropping, blurring, changing brightness, rotation...etc. And, operations at video level as duplication of frames.

Finally, we have analyzed deeply the state-of-the-art TSN. It has marked an inflexion

point in the task of action recognition. Instead of obtaining multiple frames for each video, by randomly extracting three of them evenly spaced, in conjunction with their optical flow, a network can robustly learn the representation of each action category present in the dataset. Furthermore, we have deployed and tested the TSN pre-trained with the UCF-101 with self recorded data. Specifically, we have studied the output by passing a video sequence of preparing a coffee cup.

5.2. Future Work

Due to time constraint imposed on this project and the unexpected pandemic COVID-19, many possible improvements and ideas were left out for future works. Here we summarize them to conclude this Thesis:

- Improve the tagging tool in order to speed up the tagging process of the videos. One possible improvement is to ease the tagging by using the same instance for multiple videos, i.e., cleaning the cache of an already tagged video. Finally, as a novel approach in video tagging could be applying active learning in order to improve the results and, also, accelerate the tagging.
- Our video data augmentation is randomly applied. However, it would be interesting to study the best combination of parameters in order to obtain realistic and usefull variability in data.
- Conclude the UnrealRox extension with a procedural generation of synthetic data. The goal is to create a framework with the possibility of automatically and randomly generate data with different BVH animations, meshes and environments.
- Once generated synthetic per-frame action tagged videos, proceed with training and testing the Temporal Segment Network. In other words, search for the best possible configuration to maximize output accuracy. Also, it would be interesting to check how the network reacts with the real and synthetic mixed data.
- Adapt the Temporal Segment Network in order to support the prediction of the next behaviour, action and sub-action in the sequence. In detail, experiment the use of the TSN with the Dynamic Time Warping (DTW) algorithm. As the importance in predicting the next step is to understand the steps followed up to that moment, DTW helps to classify from a sequence. The TSN classifies in real time each moment which step is executed. When the prediction is needed, the DTW finds which sequence adjusts better to the recorded and outputs the next step.

Bibliography

[1] Samy Bakheet. "An SVM Framework for Malignant Melanoma Detection Based on Optimized HOG Features". In: *Computation* 5 (Mar. 2017), p. 4. DOI: 10.3390/computation5010004.

[2] Joao Carreira and Andrew Zisserman. *Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset*. 2017. arXiv: 1705.07750 [cs.CV].

[3] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Nießner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. *Matterport3D: Learning from RGB-D Data in Indoor Environments*. 2017. arXiv: 1709.06158 [cs.CV].

[4] Navneet Dalal, Bill Triggs, and Cordelia Schmid. "Human Detection Using Oriented Histograms of Flow and Appearance". In: *European Conference on Computer Vision (ECCV '06)*. Ed. by Ales Leonardis, Horst Bischof, and Axel Pinz. Vol. 3952. Lecture Notes in Computer Science (LNCS). Graz, Austria: Springer-Verlag, May 2006, pp. 428–441. DOI: 10.1007/11744047_33. URL: https://hal.inria.fr/inria-00548587.

[5] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. *Convolutional Two-Stream Network Fusion for Video Action Recognition*. 2016. arXiv: 1604.06573 [cs.CV].

[6] Alberto Garcia-Garcia, Pablo Martinez-Gonzalez, Sergiu Oprea, John Alejandro Castro-Vargas, Sergio Orts-Escolano, Jose Garcia-Rodriguez, and Alvaro Jover-Alvarez. *The RobotriX: An eXtremely Photorealistic and Very-Large-Scale Indoor Dataset of Sequences with Robot Trajectories and Interactions*. 2019. arXiv: 1901.06514 [cs.CV].

[7] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. "Large-scale Video Classification with Convolutional Neural Networks". In: *CVPR*. 2014.

[8] Eric Kolve et al. *AI2-THOR: An Interactive 3D Environment for Visual AI*. 2017. arXiv: 1712.05474 [cs.CV].

[9] Hilde Kuehne, Hueihan Jhuang, Estibaliz Garrote, Tomaso Poggio, and Thomas Serre. "HMDB51: A Large Video Database for Human Motion Recognition". In: Nov. 2011, pp. 2556–2563. DOI: 10.1109/ICCV.2011.6126543.

[10] Laptev and Lindeberg. "Space-time interest points". In: *Proceedings Ninth IEEE International Conference on Computer Vision*. Oct. 2003, 432–439 vol.1. DOI: 10.1109/ICCV.2003.1238378.

[11] Pablo Martinez-Gonzalez, Sergiu Oprea, Alberto Garcia-Garcia, Alvaro Jover-Alvarez, Sergio Orts-Escolano, and Jose Garcia-Rodriguez. *UnrealROX: An eXtremely Photorealistic Virtual Reality Environment for Robotics Simulations and Synthetic Data Generation*. 2018. arXiv: 1810.06936 [cs.RO].

[12] Luiza Mici, German I. Parisi, and Stefan Wermter. "Compositional Learning of Human Activities With a Self-Organizing Neural Architecture". In: *Frontiers in Robotics and AI* 6 (2019), p. 72. ISSN: 2296-9144. DOI: 10.3389/frobt.2019.00072. URL: https://www.frontiersin.org/article/10.3389/frobt.2019.00072.

[13] Xiaojiang Peng, Limin Wang, Xingxing Wang, and Yu Qiao. *Bag of Visual Words and Fusion Methods for Action Recognition: Comprehensive Study and Good Practice*. 2014. arXiv: 1405.4506 [cs.CV].

[14] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. *VirtualHome: Simulating Household Activities via Programs*. 2018. arXiv: 1806.07011 [cs.CV].

[15] Manolis Savva, Angel X. Chang, Alexey Dosovitskiy, Thomas Funkhouser, and Vladlen Koltun. *MINOS: Multimodal Indoor Simulator for Navigation in Complex Environments*. 2017. arXiv: 1712.03931 [cs.LG].

[16] Manolis Savva et al. *Habitat: A Platform for Embodied AI Research*. 2019. arXiv: 1904.01201 [cs.CV].

[17] Karen Simonyan and Andrew Zisserman. *Two-Stream Convolutional Networks for Action Recognition in Videos*. 2014. arXiv: 1406.2199 [cs.CV].

[18] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. *UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild*. 2012. arXiv: 1212.0402 [cs.CV].

[19] César Roberto de Souza, Adrien Gaidon, Yohann Cabon, Naila Murray, and Antonio Manuel López. *Generating Human Action Videos by Coupling 3D Game Engines and Probabilistic Graphical Models*. 2019. arXiv: 1910.06699 [cs.CV].

[20] Julian Straub et al. *The Replica Dataset: A Digital Replica of Indoor Spaces*. 2019. arXiv: 1906.05797 [cs.CV].

[21] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. *Learning Spatiotemporal Features with 3D Convolutional Networks*. 2014. arXiv: 1412.0767 [cs.CV].

[22] Heng Wang, Alexander Kläser, Cordelia Schmid, and Cheng-Lin Liu. "Dense Trajectories and Motion Boundary Descriptors for Action Recognition". In: *International Journal of Computer Vision* 103 (May 2013). DOI: 10.1007/s11263-012-0594-8.

[23] Heng Wang and Cordelia Schmid. "Action Recognition with Improved Trajectories". In: *ICCV - IEEE International Conference on Computer Vision*. Sydney, Australia: IEEE, Dec. 2013, pp. 3551–3558. DOI: 10.1109/ICCV.2013.441. URL: https://hal.inria.fr/hal-00873267.

[24] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. *Temporal Segment Networks for Action Recognition in Videos*. 2017. arXiv: 1705.02953 [cs.CV].

[25] Xuanhan Wang, Lianli Gao, Peng Wang, Xiaoshuai Sun, and Xianglong Liu. "Two-Stream 3-D convNet Fusion for Action Recognition in Videos With Arbitrary Size and Length". In: *IEEE Transactions on Multimedia* PP (Sept. 2017), pp. 1–1. DOI: 10.1109/TMM.2017.2749159.

[26] Fei Xia, Amir Zamir, Zhi-Yang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. *Gibson Env: Real-World Perception for Embodied Agents*. 2018. arXiv: 1808.10654 [cs.AI].

[27] Claudia Yan, Dipendra Misra, Andrew Bennnett, Aaron Walsman, Yonatan Bisk, and Yoav Artzi. *CHALET: Cornell House Agent Learning Environment*. 2018. arXiv: 1801.07357 [cs.AI].

A. Appendix I

A.1. Recorded behaviours

Figure A.1: Behaviour of preparing a glass of water.

Figure A.2: Behaviour of preparing a coffee cup.

Figure A.3: Behaviour of opening a door to a colleague.

A.2. Comparison with real and synthetic data

Figure A.4: Comparison of real recorded data and generated synthetic data.